Obserwowana na polskim rynku inwazja mikrokomputerów kompatybilnych z IBM PC uzmysłowiła wielu konstruktorom, że już najwyższa pora zająć się konstrukcjami 16-bitowymi. Właśnie kariera komputera osobistego wypuszczonego przez informatycznego kolosa powoduje, że mikroprocesor 8086 ma olbrzymią szansę zdominować wszelkie próby podejmowane przez projektantów. Narzucana konstruktorom zatrudnionym w firmach państwowych konieczność opierania się na podzespołach produkowanych w krajach RWPG sprawiła, że wobec zapowiedzi z ZSRR, pierwsze wprowadzane do produkcji mikrokomputery 16-bitowe opierają się właśnie na 8086. W poniższym tekście nie napisano, dlaczego w krajach przodujących nie stosuje się już 8086 w projektowanych konstrukcjach. Wymagałoby to bowiem szczegółowego porównania z nowszymi mikroprocesorami (np. 68000, 80186, 80286 i in.). Lawina już ruszyła i mikroKLAN jej nie powstrzyma, na nic więc zdałoby się wykazywanie, że istnieją lepsze układy. Skoro jednak 8086 będzie wykorzystywany w krajowych konstrukcjach, warto poświęcić mu trochę miejsca. Poniższe uwagi powinny ułatwić opracowywanie koncepcji przyszłych systemów w taki sposób, by „cienie" były jak najmniej bolesne, a „blaski" posuwały nas choć trochę do przodu.

Blaski i cienie mikroprocesora 8086

Mikroprocesor 8086 jest jednym z pierwszych mikroprocesorów 16-bitowych. W porównaniu z wcześniejszymi mikroprocesorami 8-bitowymi cechuje go znacznie większa moc przetwarzania, stwarzająca możliwość wykorzystywania w zastosowaniach, które przedtem były realizowane w oparciu o minikomputery.

Jednak, jak każde pionierskie rozwiązanie, zawiera on wiele niefortunnych cech, utrudniających jego wykorzystanie. W kolejnych układach mikroprocesorów „spokrewnionych" z 8086 — a więc układach 80186 i 80286 — niektóre z tych błędów starano się wyeliminować.

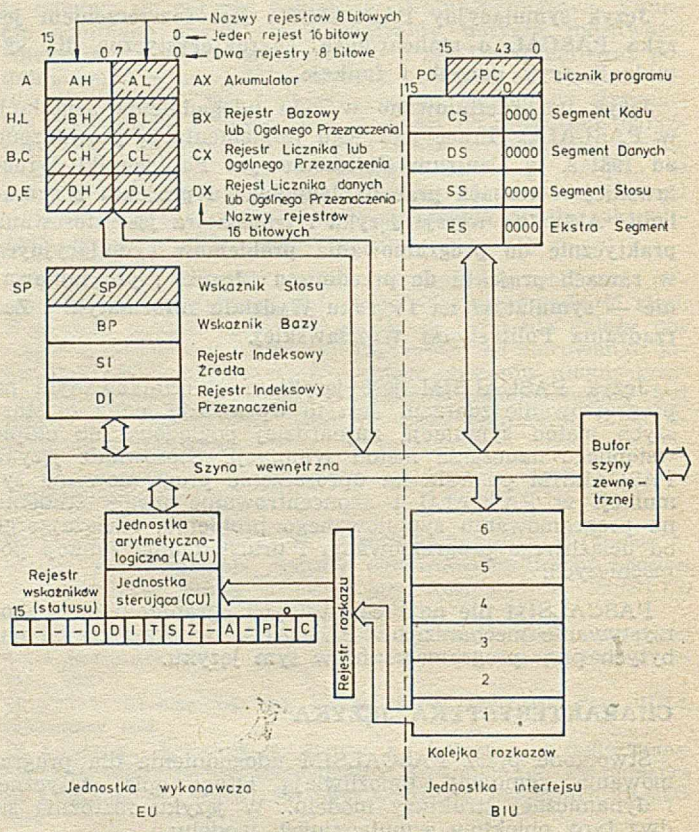
Mikroprocesor 8086 ma identyczną architekturę jak 8088. Różnica polega jedynie na szerokości szyny danych, która dla mikroprocesora 8086 jest 16-bitowa, a dla 8088 — 8-bitowa. Zastosowanie 8-bitowej szyny danych upraszcza budowę systemu, ale kosztem wydłużenia czasu, w jakim realizowane są wszelkie operacje zapisu i odczytu. Dalsze uwagi o mikroprocesorze 8086 można w równym stopniu odnosić do układu 8088.

Dość istotną cechą 8086 jest pewne podobieństwo do popularnego 8080. Szeroko reklamowana jest m. in. kompatybilność języka ASSEMBLER 8086 z językiem ASSEMBLER 8080. Ma to oznaczać, że zestaw rozkazów mikroprocesora 8080 jest podzbiorem zestawu rozkazów 8086. Kompatybilność ograniczona jest jednak do postaci źródłowej programów. Kod maszynowy 8080 nie może być bezpośrednio wykorzystany przez mikroprocesor 8086. Również kompatybilność na poziomie wersji źródłowych programów nie jest pełna. Mnemoniki języka ASSEMBLER 8086 są w przeważającej części nieco inne od stosowanych dla analogicznych operacji w języku ASSEMBLER 8080. Tym niemniej koncepcja logiczna obu języków jest identyczna. W rezultacie zapis symboliczny programu w języku ASSEMBLER 8086 jest podobnie mało przejrzysty jak dla 8080. Jako przykład języka symbolicznego bardziej przejrzystego i prostszego do opanowania pamięciowego można podać ASSEMBLER Z80.

Zestaw rozkazów 8080 nie jest w pełni podzbiorem języka ASSEMBLER 8086. Przykładowo, rozkazy warunkowego powrotu z podprogramu w przypadku 8086 muszą zostać zastąpione przez dwa rozkazy: skoku warunkowego i powrotu.

Programy ASSEMBLER dla 8086 są zazwyczaj tak skonstruowane, że rozpoznają również mnemoniki ASSEMBLERA 8080 i generują kod wynikowy odpowiedniego rozkazu 8086 (lub kilku rozkazów).

Podstawową nowością w strukturze 8086 jest oczywiście możliwość wykonywania operacji na słowach o długości 16 bitów. Dla zapewnienia kompatybilności z 8080 wprowadzono jednak możliwość podziału rejestrów ogólnego zastosowania na 8-bitowe części. Na rysunku 1 pokazano zestaw rejestrów wewnętrznych 8086.



Rys. 1. Architektura mikroprocesora 8086/88. Na rysunku wydzielono linią przerywaną bloki funkcjonalne EU i BIU. Znaczenie poszczególnych bitów w rejestrze wskaźników:
0 — przekroczenie zakresu, D — kierunek transferu, I — przerwania odblokowane/zablokowane, T — pułapka, S — znak, Z — zero, A — przeniesienie pośrednie, P — parzystość, C — przeniesienie

Projektanci 8086 wydzielili dwa bloki funkcjonalne tworząc w strukturze mikroprocesora jak gdyby dwa „podprocesory" (rys. 1). Jeden z nich nazwany został jednostką interfejsu BIU (ang. Bus Interface Unit). BIU jest odpowiedzialny za wszelkie kontakty ze światem zewnętrznym. Drugi to jednostka wykonawcza EU (ang. Execution Unit). Rozpoznaje ona pobrane z pamięci przez BIU rozkazy i steruje ich wykonaniem. Obie części współpracują ze sobą w sposób asynchroniczny na zasadzie przekazywania zleceń i potwierdzeń.

Jednostka BIU wprowadza do 6-bitowej kolejki kody i argumenty kolejnych rozkazów aż do jej zapełnienia. W tym samym czasie jednostka EU zajęta jest zdekodowaniem i realizacją wcześniej wprowadzonego rozkazu. Jeżeli EU stwierdzi konieczność wykonania dodatkowej operacji odczytu lub zapisu, wystawia odpowiednie zlecenie dla BIU, które je realizuje natychmiast po zakończe-

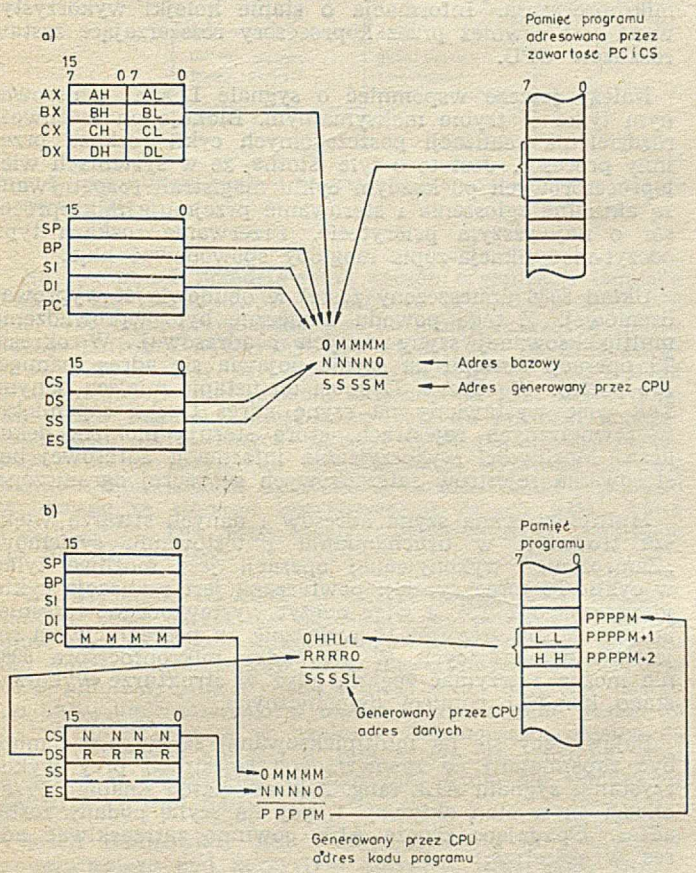
niu aktualnie wykonywanego cyklu. Przyjęcie takiego rozwiązania w znacznym stopniu eliminuje czas martwy, w którym CPU oczekuje na wprowadzenie kodu rozkazu.

Jeżeli w programie wystąpi rozkaz skoku, to kolejka będzie ładowana od początku, a EU musi zaczekać do chwili aż zostanie wprowadzony kod pierwszej operacji z nowego obszaru. Podobnie, odwołanie do informacji, którą trzeba odczytać z pamięci, wstrzymuje pracę EU do momentu aż BIU zrealizuje zlecenie.

BIU przechodzi w stan pasywny tylko wtedy, gdy zapełniona jest kolejka i nie ma żadnych zleceń od EU. Ogranicza to możliwość stosowania tzw. przezroczystych transmisji DMA lub cykli odświeżających pamięci dynamiczne. Polegają one na podkradaniu cykli (ang. cycle stealing), w których CPU zajęte jest dekodowaniem rozkazu i nie realizuje operacji dostępu do pamięci. W ten sposób transfer informacji między urządzeniami wejścia-wyjścia a pamięcią lub cykl odświeżania pamięci dynamicznych nie zakłóca realizacji programu.

W przypadku mikroprocesora 8086 pasywne cykle BIU występują w sposób nieregularny, co praktycznie eliminuje możliwość wykorzystania ich przy transmisji DMA lub do odświeżenia pamięci. Ponieważ jednak i dla wcześniejszych mikroprocesorów firmy INTEL realizacja wspomnianych rozwiązań nie była łatwa, należy uznać, że wprowadzenie kolejki rozkazów jest ulepszeniem zwiększającym efektywność pracy systemu. Typowo, realizacja rozkazu wymaga od dwóch do trzydziestu cykli zegarowych.

Projektując mikroprocesor mający konkurować z minikomputerami, konstruktorzy musieli zwiększyć obszar bezpośrednio dostępnej pamięci (tzn. takiej, którą mikroprocesor może zaadresować bez potrzeby przełączania sprzętowo zrealizowanych stron). Mikroprocesor 8086 może bezpośrednio wykorzystywać do 1 megabajta pamięci.



Rys. 2. Metoda wyznaczania adresu fizycznego przy operacji dostępu do pamięci

a) rejestry, które mogą być uwzględnione w operacjach na danych

b) przykład wyznaczania adresu z uwzględnieniem wartości podanej bezpośrednio w kodzie programu

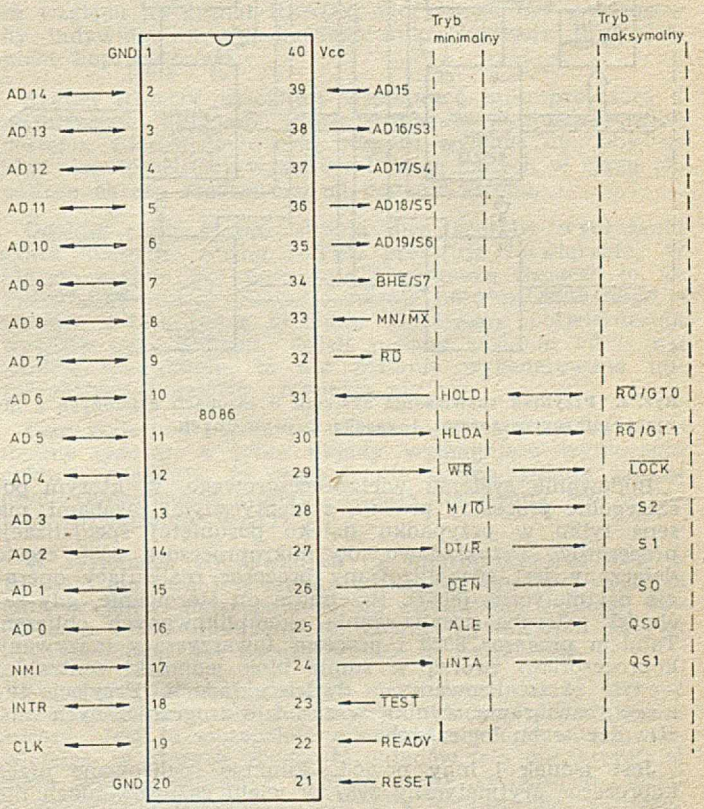
Wszelkie operacje dostępu do pamięci odbywają się za pośrednictwem rejestru segmentu pamięci (rys. 2). Dzięki takiemu rozwiązaniu wszystkie programy są relokowalne. Oznacza to, że kod programu można przepisywać w dowolne miejsca pamięci i zlecać jego realizację CPU, bez potrzeby wprowadzania modyfikacji.

Możliwe jest także wykorzystywanie trybu adresowego, w którym adresy fizyczne obliczane są z uwzględnieniem bieżącej wartości rejestru SI. Napisane w ten sposób procedury wielobieżne (ang. reentrant) mogą zostać ponownie wywołane przez wstrzymujące ich realizację zewnętrzne przerwanie bez obawy zamazania wcześniej zapisanych informacji. Procedura może też wywoływać samą siebie.

Rozbudowane metody generowania fizycznych adresów pamięci są świetnym narzędziem w rękach doświadczonego programisty. Równocześnie jednak tworzą trudną do sforsowania barierę dla mniej wprawnych osób. Dlatego też znaczna część programistów preferuje stosowanie języków wyższego poziomu. Sprzyja to zwiększeniu przejrzystości pisanych przez nich programów, a także sprawia, że finalny produkt często ma efektywność działania zbliżoną do programów pisanych w języku ASSEMBLER dla mikroprocesorów 8-bitowych. Można stwierdzić, że większa moc przetwarzania mikroprocesora 8086 jest wykorzystywana w dużej mierze dla polepszenia komfortu pracy programistów.

Mikroprocesor 8086 może działać w dwóch różnych trybach: minimalnym oraz maksymalnym (rys. 3). Wybór trybu pracy nie ma wpływu na realizację programu, zmienia natomiast sposób współpracy z otoczeniem. Tryb wybierany jest przez przyłączenie wyprowadzenia MN-MX/ do masy (tryb maksymalny) lub +5 V (tryb minimalny).

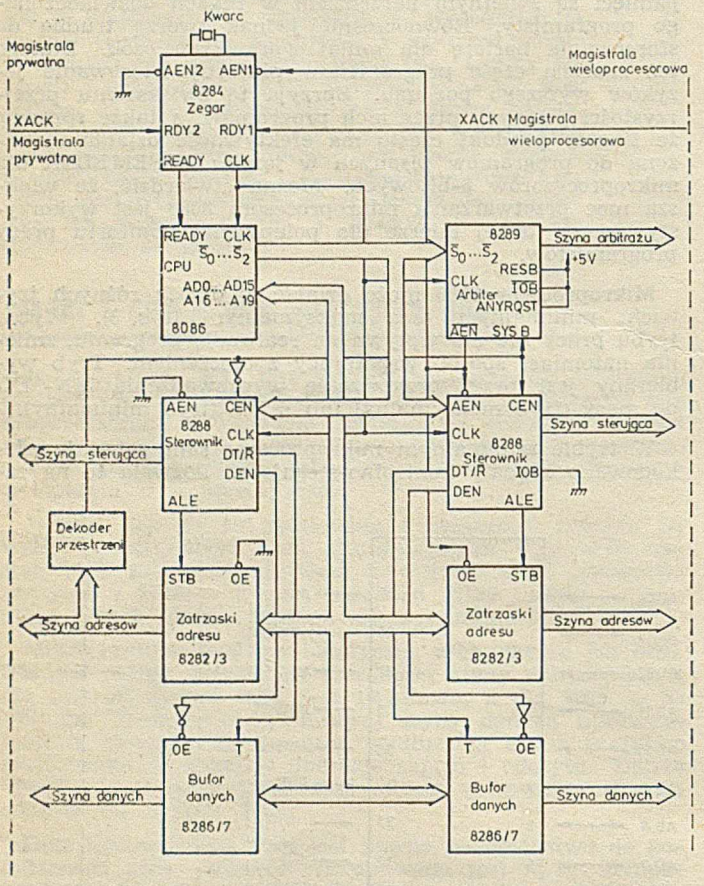
W trybie minimalnym mikroprocesor sam generuje zdekodowane sygnały kontrolno-sterujące. Pozwala to na za-



Rys. 3. Topografia wyprowadzeń układu 8086 z uwzględnieniem dwóch trybów pracy: minimalnego i maksymalnego

oszczędzenie dodatkowego układu sterującego szyną. Wykorzystanie trybu minimalnego ogranicza się do systemów jednoprocesorowych. W trybie tym jedynie układ DMA może na pewien czas przejąć sterowanie magistralami.

W trybie maksymalnym 8086 generuje jedynie sygnały statusu informujące o typie wykonywanej operacji. Dzięki temu możliwe jest zastosowanie dwóch (lub więcej) układów sterujących różnymi szynami. W ten sposób pamięć i układy wejścia-wyjścia można rozdzielić na zasoby prywatne mikroprocesora i zasoby dostępne również dla innych mikroprocesorów pracujących w systemie (rys. 4). Możliwość wprowadzenia takiego podziału w systemie wieloprocesorowym jest podstawowym warunkiem efektywnej pracy, gdyż w danym momencie z magistrali systemowej może korzystać tylko jeden procesor. Pozostałe muszą czekać — chyba że dysponują prywatną pamięcią, z której odczytują program bez wchodzenia w kolizję z mikroprocesorem wykorzystującym magistralę systemową.



Rys. 4. Przykład rozwiązania systemu o zasobach dzielonych z innymi mikroprocesorami i zasobach prywatnych

Budowanie systemu wieloprocesorowego, w którym poszczególne procesory pracują z identycznymi zasobami, ma sens tylko w przypadku daleko posuniętej specjalizacji procesorów. Przykładowo, do mikroprocesora 8086 często dołączany jest specjalizowany procesor realizujący operacje arytmetyczne (8087). Przejmuje on sterowanie, gdy zachodzi potrzeba zrealizowania skomplikowanych obliczeń. Tandem procesor 8086 i procesor towarzyszący (nazywany koprocesorem) tworzą w sumie blok jednostki centralnej — tyle że zrealizowany na dwóch układach. Przyjęcie takiego rozwiązania wynika w zasadzie z ograniczonych możliwości technologicznych.

Jest jednak i inny powód. Funkcje realizowane przez koprocesor arytmetyczny 8087 w wielu zastosowaniach nie są potrzebne, natomiast przydatne mogłyby się okazać zupełnie inne. Wychodząc naprzeciw różnym potrzebom użytkowników firma INTEL przygotowuje całą rodzinę specjalizowanych koprocesorów. Obecnie oferowany jest już koprocesor LAN 82586 (ang. Local Area Network — układ realizujący funkcje związane z obsługą sieci lokalnych), koprocesor tekstu 82730 (dedykowany zagadnieniom związanym z przetwarzaniem plików tekstowych) i koprocesor wejścia-wyjścia 8089. Ostatni z wymienionych może być również wykorzystywany jako samodzielny procesor specjalizujących się w realizacji funkcji związanych z obsługą kanałów wejścia-wyjścia.

Dołączenie jednego z koprocesorów nie wyklucza zastosowania następnego. Możliwe jest dzięki temu komponowanie jednostek centralnych o dużej mocy przetwarzania, dostosowanych do obsługi konkretnego zagadnienia.

W trybie mnimalnym dla przejęcia sterowania przez układ DMA wykorzystywane są sygnały HOLD i HLDA. W trybie maksymalnym założono, że sterowanie będzie przejmowane jedynie przez koprocesory. Dlatego też odstąpiono od tradycyjnego protokołu wprowadzając w zamian wymianę żądania i potwierdzenia (RQ/—GT/) przesyłane po tej samej linii. Układ, który chce przejąć sterowanie, zgłasza ten zamiar generując impuls RQ/. Z chwilą zakończenia aktualnie wykonywanej operacji, 8086 generuje impuls potwierdzenia GT/ i zwalnia szynę.

Przyjęte rozwiązanie umożliwia prostą realizację współpracy nawet z większą liczbą koprocesorów (dwie linie o różnych priorytetach, możliwość kaskadowego przekazywania żądań i potwierdzeń). Natomiast dołączenie tradycyjnego układu DMA (np. 8257, 8237) do działającego w trybie maksymalnym 8086 wymaga zrealizowania stosunkowo złożonej konstrukcji układowej.

W trybie maksymalnym możliwy jest odczyt statusu kolejki rozkazów (wyprowadzenia QS0 i QS1). Pozwala to wykorzystać zewnętrzne urządzenia do śledzenia pracy mikroprocesora. Informacja o stanie kolejki wykorzystywana jest również przez koprocesory rozszerzające zestaw rozkazów CPU.

Należy jeszcze wspomnieć o sygnale LOCK/ generowanym tylko w trybie maksymalnym. Blokuje on możliwość rozdzielenia realizacji poszczególnych cykli rozkazu przez inny procesor. Jest to o tyle istotne, że w systemach wieloprocesorowych po każdym cyklu magistrali rozpatrywane są aktualne zgłoszenia i sterowanie przejmuje mikroprocesor o najwyższym priorytecie. Przerwanie rozkazu typu odczyt-modyfikacja-zapis mogłoby spowodować błąd.

Układ 8086 umieszczony został w obudowie 40-wyprowadzeniowej. Z tego powodu konieczne było wprowadzenie multipleksowanej szyny danych i adresowej. W okresie T1 operacji dostępu na szynie pojawia się adres (pamięci lub portu). W okresie T2 musi on ustąpić miejsca danym. Tak więc wystawiony na szynę adres trzeba zapamiętać w zewnętrznych rejestrach, które sterują blokiem dekodera. Możliwości wykorzystania informacji adresowej bez stosowania rejestrów zatrzaskowych są raczej ograniczone.

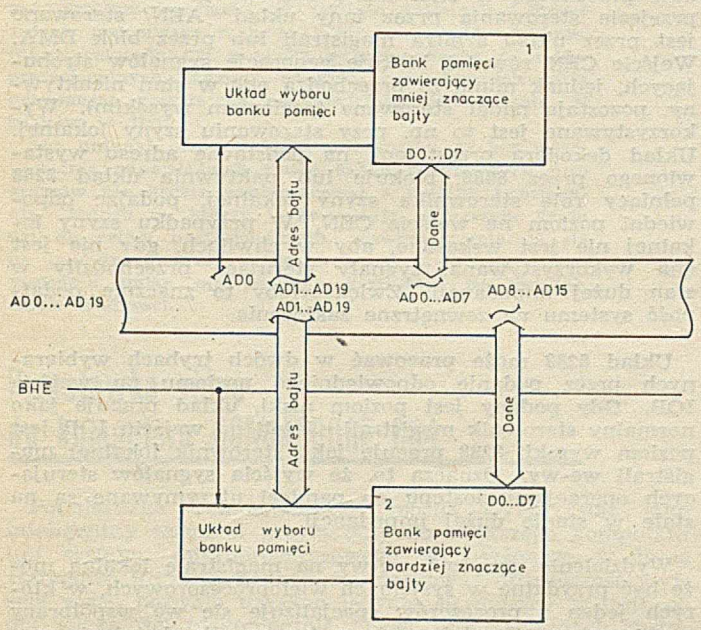
Multipleksowana szyna adresów i danych stwarza większe trudności w uruchamianiu i testowaniu systemów. „Zamrożenie" wykonywanej operacji jest możliwe tylko w cyklu T3 (precyzyjniej: powtarzana jest realizacja cyklu identycznego z T3), a więc odczyt wystawionego wcześniej adresu można zrealizować wyłącznie za pośrednictwem rejestrów pamiętających. W przypadku mikroprocesora 8086 nie można zatrzymać zegara, gdyż w strukturze wykorzystano komórki pamięci dynamicznych.

Pojawiający się na multipleksowanej szynie adres może być zapamiętany w zewnętrznych rejestrach przy wykorzystaniu sygnału ALE (ang. Address Latch Enable). Przechodzi on w stan aktywny 1, gdy na szynę podany został adres. Opadające zbocze ALE powinno zatrzaskiwać adres w rejestrze.

Dodatkową trudnością wynikającą z multipleksowanej szyny adresów i danych jest konieczność zagwarantowania, że dane z pamięci lub układu wejścia-wyjścia nie będą zakłócać wystawianych adresów. Dlatego obwody wyjściowe współpracujących układów muszą być uaktywniane nie przez sygnał z dekodera adresów, lecz przez

strob odczytu MRDC/ lub IORC/. Układy produkowane przez firmę INTEL mają niezależne wejścia CS/ i RD/, co eliminuje możliwość konfliktu.

Mikroprocesor 8088 przesyła dane jedynie w postaci bajtowej. 8086 przesyła słowa (16 bitów) oraz bajty. Przesłania bajtowe wprowadzono dlatego, że kody niektórych rozkazów składają się z nieparzystej liczby bajtów. W ten sposób programy zajmują mniej miejsca, lecz organizacja pamięci jest bardziej złożona, przez co komplikują się również układy dekoderów. Przyjęte rozwiązanie wydaje się obecnie dość kontrowersyjne. Układy pamięci są już stosunkowo tanie, natomiast oszczędność miejsca jest raczej niewielka.



Rys. 5. Organizacja pamięci w systemie sterowanym przez mikroprocesor 8086

Na rysunku 5 przedstawiono sposób organizacji pamięci. Składa się ona z dwóch banków o organizacji bajtowej. Pierwszy bank przyłączony jest do mniej znaczącego bajtu szyny danych i wybierany jest wtedy, gdy na linię adresową AD0 podany zostanie stan logiczny 0. Drugi bank wybierany jest przez poziom aktywny na linii BHE/ (ang. Bus High Enable) i przyłączony jest do bardziej znaczącego bajtu szyny danych. Odczyt słowa (16-bitowy) następuje tylko w przypadku równoczesnego wybrania obydwu banków. Poszczególne komórki w ramach banków pamięci wybierane są stanami logicznymi na liniach AD1 ... AD15 i A16 ... A19. Tak więc każdy z banków może pomieścić do 512 KB pamięci.

W przyjętym rozwiązaniu odczyt słowa może nastąpić jedynie spod parzystego adresu (tylko wtedy AD0 przyjmuje stan 0). Jeżeli wykonywany rozkaz wymaga dostępu do słowa umieszczonego pod nieparzystym adresem, to CPU automatycznie wykonuje dwa cykle: najpierw dostęp do bajtu umieszczonego w banku 2, a następnie dostęp do bajtu umieszczonego w banku 1. Równoczesny dostęp do obydwu banków nie jest w tym przypadku możliwy, gdyż komórki w poszczególnych bankach różnią się stanem logicznym na linii AD1.

Jeżeli realizowany jest dostęp do bajtu (a nie słowa) umieszczonego pod parzystym adresem, linia BHE/ przyjmuje stan pasywny 1. Podobnie, przy dostępie do bajtu umieszczonego pod nieparzystym adresem, linia AD0 jest w stanie 1. Warto zwrócić uwagę, że przy ładowaniu słowa umieszczonego pod nieparzystym adresem do rejestru 16-bitowego wykonane zostaną dwa cykle odczytu. Najpierw wprowadzony zostanie bajt z banku dołączonego do linii AD8 ... AD15 szyny danych. Przeznaczony jest on dla mniej znaczącego bajtu rejestru 16-bitowego. Drugi odczyt nastąpi z banku przyłączonego do lini AD0 ... AD7 i trafi do bardziej znaczącego bajtu rejestru.

Ten pozornie zagmatwany sposób dostępu staje się bardziej przejrzysty, jeżeli linię adresową AD0 potraktować podobnie jak BHE/ — czyli strob wyboru banku. Wtedy tylko AD1 ... AD15 są liniami wyznaczającymi adres komórki. Poszczególne bajty składające się w słowa umieszczane są na przemian w obydwu bankach.

Linie adresowe A16 ... A19 nie są multipleksowane z danymi, lecz z informacjami o charakterze statusu. A16 i A17 informują o segmencie bazowym, który wykorzystywany jest w aktualnie realizowanej operacji. Linia A18 odzwierciedla stan wskaźnika informującego, czy możliwe jest przyjmowanie przerwań. Linia A19 utrzymywana jest w stanie 0, gdy mikroprocesor steruje szyną.

Dość specyficzną rolę spełnia wyprowadzenie o nazwie TEST/. Jest ono wykorzystywane w połączeniu z rozkazem WAIT, który wprowadza CPU w stan oczekiwania na pojawienie się poziomu aktywnego na wejściu TEST/. Podobną operację można by w zasadzie zrealizować wykorzystując przerwania. Jednak wejście TEST/ można wykorzystać także wtedy, gdy przerwania są zablokowane lub gdy 8086 przekazał sterowanie do koprocesora (znajdujące się w kolejce rozkazy realizowane są dopóki BIU nie otrzyma zlecenia wymagającego dostępu poprzez szynę).

UKŁAD ZEGARA 8224

Konstrukcja oparta na mikroprocesorze 8086 musi zawierać, podobnie jak w przypadku mikroprocesora 8080, dodatkowy układ zegara. Układ 8284, poza generacją sygnału zegarowego taktującego pracą mikroprocesora, wykorzystywany jest do synchronizacji niektórych sygnałów kontrolnych (np. RESET, READY).

W systemach wieloprocesorowych mogłoby wydawać się celowe zastosowanie jednego układu zegara 8284 dla wszystkich mikroprocesorów. Nie jest to jednak możliwe ze względu na sygnał READY, który musi być generowany indywidualnie dla każdego mikroprocesora (z wyjątkiem koprocesorów).

Sygnał READY umożliwa współpracę mikroprocesora z wolniejszymi układami, które wymagają wydłużenia cyklu dostępu. Standardowy cykl odczytu wymaga np., aby układ współpracujący wysterował szynę danych w ciągu ok. półtora okresu zegarowego od podania strobu.

Ogólnie rzecz biorąc istnieją dwa sposoby wydłużania cyklu operacji. Firma INTEL preferuje rozwiązanie, w którym system jest normalnie niegotowy. Oznacza to, że każdy zaadresowany układ generuje potwierdzenie umożliwiające dalszą pracę systemu. Opóźnienie potwierdzenia wydłuża cykl dostępu. Jeżeli podany zostanie adres nie istniejącego układu, to nie zostanie wygenerowane potwierdzenie i system „zawiesza się".

Inne podejście polega na założeniu, że system jest normalnie gotowy, a tylko układy wymagające wydłużenia cyklu sygnalizują, że operacji nie można jeszcze zakończyć. Odpowiednik sygnału READY nazywany jest w takim rozwiązaniu WAIT/.

W systemach zbudowanych na mikroprocesorze 8086 w zasadzie możliwe jest zrealizowanie obu koncepcji. Jednak czas, w jakim musi pojawić się ewentualne żądanie wydłużenia cyklu (mierzony od chwili wystawienia aktywnych strobów), jest bardzo trudny do osiągnięcia.

Do wad systemu normalnie niegotowego należy zaliczyć konieczność znacznej rozbudowy układów dekodujących adresy. Jest to szczególnie odczuwalne w systemach niewielkich, w których większość układów (ale nie wszystkie) może współpracować z CPU z pełną szybkością. Jeżeli z określonych powodów nie zostały umieszczone w zbliżonym obszarze przestrzeni adresowej „gotowe" układy, to każdy z nich należy rozbudować o blok generacji potwierdzenia READY.

Jeżeli sygnał READY generowany jest po szczegółowym określeniu komórki przez dekoder, to może on dotrzeć do CPU zbyt późno i cykl dostępu zostaje niepotrzebnie wydłużony. Dlatego też niedawno wprowadzono nową wersję układu zegarowego 8284A. Umożliwia on wykorzystanie zewnętrznych układów TTL do synchronizacji READY z zegarem sterującym pracą CPU.

Dla ochrony przed skutkami zablokowania systemu przez zaadresowanie nieistniejącej komórki, warto wprowadzić specjalny układ czasowy zwany budzikiem (ang. watch dog timer), który po zbyt długim braku potwierdzenia generuje je sam, sygnalizując ewentualnie błędną sytuację zgłoszeniem przerwania.

Systemy normalnie gotowe wymagają bardzo szybkiego stwierdzenia, że zaadresowany został układ wymagający wydłużenia cyklu. Utrudnia to realizację rozbudowanych systemów, szczególnie systemów wieloprocesorowych.

Podsumowując, 8086 predestynowany jest do pracy w rozbudowanych systemach skonstruowanych jako normalnie niegotowe lub w systemach, gdzie nie zachodzi potrzeba wydłużania cyklu operacji (w tym przypadku system jest stale gotowy). Również praca 8086 w trybie minimalnym nie zmienia powyższej konkluzji, gdyż czas pozostawiony na ewentualne zgłoszenie WAIT/ jest jeszcze krótszy. Jest to wyraźna wada, gdyż tryb minimalny miał upraszczać konstruowanie niewielkich systemów.

READY powstaje ze zsumowania sygnałów z wielu źródeł. Dla uproszczenia rozwiązań układowych w 8284 przewidziano cztery wejścia dla obsługi tego zgłoszenia. Możliwe jest sumowanie sygnałów z dwóch źródeł, przy czym w zależności od polaryzacji sygnału można wykorzystać albo wejście AEN/ albo RDY. Wykorzystanie dwóch wejść dla danego źródła istotne jest w przypadku współpracy z magistralą systemu wieloprocesorowego dla odcięcia potwierdzeń przychodzących w czasie, gdy procesor nie steruje tą magistralą. Najczęściej zbuforowana inwerterem Schmitta systemowa linia potwierdzeń XACK/ doprowadzona jest do wejścia RDY. Na wejście AEN/ podawany jest sygnał z układu arbitra magistrali (8289) określający, czy dany mikroprocesor uzyskał dostęp do magistrali systemowej.

Podstawowym zadaniem układu 8284 jest generacja sygnału zegarowego sterującego mikroprocesorem. Jego częstotliwość musi być większa od 2 MHz. Maksymalna częstotliwość zależy od wersji mikroprocesora — standardowo wynosi ona 5 MHz. Układ może generować sygnał zegarowy w oparciu o przyłączony kwarc lub sygnał zewnętrzny (zarówno kwarc, jak i przebieg zewnętrzny muszą mieć częstotliwość trzykrotnie większą od zamierzonej częstotliwości dla 8086). Wykorzystanie zewnętrznego źródła może być przydatne do synchronizacji przebiegów zegarowych w systemach wieloprocesorowych. Przesunięcia poszczególnych sygnałów zegarowych nie powinny wtedy przekroczyć 45 ns.

Układ 8284 zawiera wewnętrzny dzielnik częstotliwości przez trzy, którego stan po włączeniu zasilania nie jest zdeterminowany. Z tego powodu wprowadzono dodatkowe wejście synchronizujące CSYNC, umożliwiające zainicjowanie pracy zegara przez zewnętrzny impuls.

Układ 8284 ma trzy wyjścia sygnałów zegarowych. Wyjście OSC generuje przebieg o częstotliwości podłączonego kwarcu i może być wykorzystane jako źródło sygnału synchronizującego inne układy 8284. Wyjście CLK dostarcza przebieg sterujący mikroprocesorem. Wreszcie wyjście PCLK generuje przebieg o częstotliwości dwukrotnie mniejszej od CLK. Zegar ten wykorzystywany jest zazwyczaj przez pracujące w systemie liczniki.

UKŁAD STEROWNIKA SZYNY 8288

Układ sterownika szyny 8288 wykorzystywany jest w systemach, w których 8086 działa w trybie maksymalnym (MN-MX/ przyłączone jest do masy). Zamienia on sygnały statusu S0, S1, S2 na sygnały strobów sterujących wykonywanymi w systemie operacjami. Tak proste zadanie spełniłby jednak z równym powodzeniem zwyczajny dekoder: 1 z 8. Sterownik 8288 realizuje wiele dodatkowych funkcji.

Najistotniejszą korzyścią wynikającą z zastosowania układu 8288 jest możliwość pracy w systemach o wielu magistralach. W tym celu przewidziano wejścia kontrolne AEN/ i CEN. AEN/ wykorzystywane jest przy współpracy z magistralą wieloprocesorową lub z DMA. Jeżeli na wejście AEN/ podany zostanie poziom wysoki, sygnały strobów przechodzą w stan dużej impedancji. Umożliwia to przejęcie sterowania przez inny układ. AEN/ sterowane jest przez układ arbitra magistrali lub przez blok DMA. Wejście CEN również blokuje generację sygnałów strobujących, jednak mimo że przechodzą one w stan nieaktywny, pozostają nadal sterowane (poziomem wysokim). Wykorzystywane jest to np. przy sterowaniu szyny lokalnej. Układ dekodera przestrzeni, na podstawie adresu wystawionego przez 8086, blokuje lub uaktywnia układ 8288 pełniący rolę sterownika szyny lokalnej, podając odpowiedni poziom na wejście CEN. W przypadku szyny lokalnej nie jest wskazane, aby w chwilach, gdy nie jest ona wykorzystywana, sygnały sterujące przechodziły w stan dużej impedancji. Zwiększyłoby to znacznie podatność systemu na zewnętrzne zakłócenia.

Układ 8288 może pracować w dwóch trybach wybieranych przez podanie odpowiedniego poziomu na wejście IOB. Gdy podany jest poziom niski, układ pracuje jako normalny sterownik magistrali. Jeżeli na wejściu IOB jest poziom wysoki, 8288 pracuje jako sterownik lokalnej magistrali we-wy. Oznacza to, że wyjścia sygnałów sterujących operacjami dostępu do pamięci utrzymywane są na stałe w stanie dużej impedancji.

Wydzielenie urządzeń we-wy na magistralę lokalną może być przydatne w systemach wieloprocesorowych, w których jeden z procesorów specjalizuje się we współpracy z otoczeniem. Zmniejsza się w ten sposób obciążenie magistrali systemowej. Jednak korzyści płynące z wykorzystania do tego celu specyficznego trybu pracy sterownika 8288 wydają się niewielkie. Przykładowo, nie można całkowicie zrezygnować z dekodera przestrzeni, bowiem nie przewidziano dla układu 8288 trybu pracy, w którym sterowałby on tylko operacjami związanymi z pamięcią. Niezbędne jest więc zastosowanie układu dekodującego rodzaj realizowanej operacji i ewentualnie blokującego sterownik obsługujący magistralę systemową.

Układ 8288 rozszerza zestaw sygnałów sterujących o przyspieszone (ang. advanced) stroby zapisu. Pojawiają się one w tym samym cyklu zegarowym, co stroby odczytu, a więc o ok. 1 cykl wcześniej niż standardowe stroby zapisu. Przyspieszone stroby zapisu wygodnie jest wykorzystać w układach generujących potwierdzenie operacji (sygnał READY lub XACK) dla uniknięcia zbędnych cykli oczekiwania. Nie należy ich jednak wykorzystywać do sterowania buforów szyny danych, gdyż przechodzą one w stan aktywny zanim CPU zdąży wystawić dane.

Generowany przez 8288 sygnał MCE stosowany jest w systemach o rozbudowanym bloku obsługi przerwań, wykorzystującym możliwość kaskadowego łączenia układów 8259A. W proponowanym przez producenta rozwiązaniu strob MCE otwiera przejście dla adresu generowanego przez zarządzający układ przerwań. Adres ten pozwala podporządkowanym sterownikom przerwań zorientować się, czyje żądanie zostało spełnione.

ANDRZEJ J. PIOTROWSKI

Poniższy materiał został zainspirowany przystawką opisaną w amerykańskim czasopiśmie BYTE. Zamawiając materiał poleciliśmy jednak autorowi przystosowanie konstrukcji do krajowych warunków — oczywiście tam, gdzie ma to sens. Dostrzegamy też dla opisanej konstrukcji nieco inną klasę zastosowań niż autorzy oryginalnego rozwiązania. Poszukiwali oni taniego urządzenia typu terminal, które mogliby wykorzystać do współpracy z mikrokomputerami budowanymi w oparciu o mikroprocesor 68000. Relatywnie niska cena mikrokomputera C64 w USA skłoniła ich do przyjęcia takiego właśnie rozwiązania.

Rozpatrując dostępność i cenę C64 w Polsce nie sądzimy, aby ktokolwiek w ten sposób próbował rozwiązać problem trudności w zakupie terminali. Biorąc jednak pod uwagę fakt, że wielu krajowych użytkowników zakupiło C64 z myślą o zastosowaniu do celów profesjonalnych, sądzimy, że nieco urealnimy ich wizje proponując przystawkę umożliwiającą wyświetlanie 80 znaków w linii.

Aby uniknąć nieporozumień zaznaczamy, że bez wprowadzania odpowiednich zmian żaden z firmowych programów nie będzie współpracował z opisaną konstrukcją. Jednak ci, którzy wykorzystują w naszym kraju C64 do celów profesjonalnych, zazwyczaj opierają się na napisanych przez siebie programach, tak że wprowadzenie zmian nie powinno nastręczyć im zasadniczych trudności.

Przystawka video do komputera COMMODORE C64

Popularnego Commodora C64 można wykorzystać jako inteligentny terminal przyłączany do większego komputera lub systemu mikroprocesorowego wyposażonego w łącze szeregowe RS 232C. Jednak standardowa rozdzielczość wyjścia video pozwala uzyskać jedynie 40 znaków w linii, co dla większości niekoniecznie nawet profesjonalnych zastosowań jest stanowczo za mało. Rozwiązaniem tego problemu może być specjalna przystawka, która umożliwi wyświetlenie 80 znaków w jednej linii. Przyłączana jest ona do portu rozszerzającego Commodora (ang. Commodore Expansion Port) i zawiera: CRTC 6845, 2KB SRAM (pamięć obrazu) i 4KB EPROM (pamięć nakładki programowej i generator znaków). Idea przystawki została przedstawiona na schemacie blokowym (rys. 1).

Funkcjonowanie przystawki

Układ CRTC generuje adresy wyświetlanej na ekranie komórki pamięci obrazowej na liniach MA0 ... MA10. Adres kolejnego rzędu wyświetlanego znaku podawany jest na liniach RA0 ... RA3. W poszczególnych komórkach pamięci obrazowej zawarte są kody ASCII (American Standard Code for Information Interchange) znaków, które mają być wyświetlone na ekranie. Odczytywany z pamięci obrazowej kod znaku wykorzystywany jest do zaadresowania pamięci w generatorze znaków. Ponieważ każdy znak składa się z 8 rzędów kropek, to dla wybrania odpowiedniej komórki w pamięci generatora znaków niezbędne jest jeszcze dołączenie adresu aktualnie wyświetlanego na ekranie rzędu.

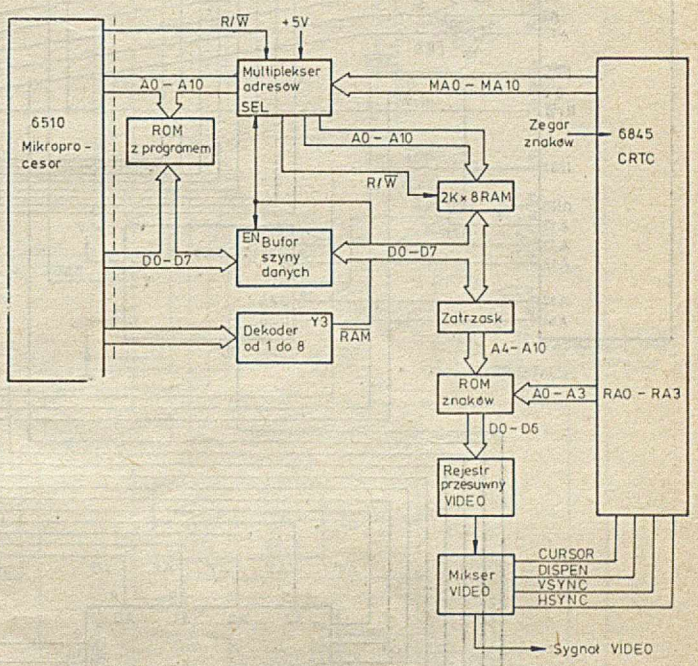
Kody odczytywane z kolejnych komórek pamięci obrazu zatrzaskiwane są na liniach adresowych pamięci ROM generatora znaków. Z CRTC podawany jest dodatkowo adres związany z aktualnie wyświetlanym rzędem. Kombinacja kodu ASCII i adresu rzędu formuje adres komórki w pamięci ROM generatora znaków, w której znajduje się odzwierciedlenie odpowiedniego rzędu dla aktualnie wyświetlanej litery. Poszczególne bity kodu znajdującego się w pamięci ROM określają, czy kolejne punkty na ekranie mają być rozjaśnione czy zaciemnione.

CRTC rozpoczyna od sekwencyjnego generowania adresów pamięci obrazowej odpowiadających 80 komórkom dla znaków z pierwszego wiersza. W tym czasie na linie adresowe rzędu podawany jest stan 0 — co oznacza, że wyświetlany jest rząd zerowy początkowych 80 znaków. Po poziomym powrocie plamki (koniec wyświetlania pierwszej linii obrazu) CRTC ponownie generuje tę samą sekwencję 80 adresów pamięci obrazu, ale tym razem z adresem rzędu równym 1 (druga linia obrazu). Cykl ten jest powtarzany do chwili aż zostaną wyświetlone wszystkie osiem rzędów dla 80 znaków z pierwszego wiersza tekstu. Następny blok 80 znaków jest generowany z adresem rzędu ustawionym znowu na zero, potem na 1 itd. Cała procedura kontynuowana jest aż do wypełnienia ekranu. Po pionowym powrocie plamki proces jest powtarzany.

Informacja z pamięci ROM generatora znaków przepisywana jest do rejestru przesuwanego, z którego sekwencyjnie (bit po bicie) podawana jest do bloku miksera video. Zadaniem miksera jest utworzenie zespolonego sygnału wizyjnego.

Na rysunku 1 pokazano również interfejs między płytką video a Commodore C64. Zarówno komputer, jak i sterownik CRT, muszą mieć dostęp do 2KB pamięci obrazowej. Układ mikroprocesora 6510 wpisuje do niej kody poszczególnych znaków ASCII, a układ 6845 steruje przesyłaniem jej zawartości do wyświetlania. W proponowanym rozwiązaniu 2KB pamięci obrazu umieszczono między adresami 9800_H ... $9FEF_H$.

Dostępem do pamięci zarządza multiplekser sterowany przez układ procesora 6510. Wystawienie przez mikroprocesor 6510 adresu z obszaru pamięci obrazu powoduje, że na wyjściu dekodera Y_3 (RAM/) pojawia się stan aktywny (logiczne zero). Sygnał ten otwiera bufor szyny danych oraz przełącza multiplekser adresowy na przekazywanie adresów i sygnału R-W/ pochodzących od mikroprocesora. Gdy 6510 nie wykonuje operacji związanej z pamięcią obrazu, adresy są wybierane przez układ 6845, a na wejście R-W/ pamięci obrazowej multiplekser dołącza sygnał +5V. W ten sposób pamięć jest utrzymywana przez cały ten okres w trybie odczytu.



Rys. 1. Schemat blokowy przystawki video do COMMODORE C 64

Układ taktujący (rys. 2) jest sercem obwodu video. Steruje on przesyłaniem wszystkich danych. Kwarc 16 MHz, trzy inwertery 7404 i dwa oporniki 620 Ω składają się na generator sygnału DOT — przebiegu prostokątnego o częstotliwości 16 MHz. Jest on wykorzystywany przez licznik 74161, czterokrotny przerzutnik typu D 74174 oraz in-

wertery do wysterowania sygnałów taktujących systemem: S-L/, CC i CC/. Gdy S-L/ jest sterowany poziomem wysokim, rejestr przesuwny przesyła dane do miksera video zgodnie z sygnałem taktującym DOT. Gdy S-L/ jest sterowany poziomem niskim, do rejestru przesuwnego wpisywane są nowe dane oraz przepisywany jest z pamięci obrazowej do układów zatrzaskowych 74174 nowy znak kodu ASCII.

Sygnał CC pełni rolę zegara znaków. Impuls generowany jest co dziewięć cykli zegara DOT. Jest to czas niezbędny do wyświetlenia jednej linii jednego znaku. Zegar znaków synchronizuje pracę sterownika CRT z resztą systemu. Sygnał CC/ jest sygnałem odwrotnym do zegara znaków. W generatorze znaków część obwodu składająca się z układów 74174 jest używana do zatrzymywania danych na liniach adresowych doprowadzonych do pamięci ROM. Opadające zbocze sygnału zegara znaków przygotowuje linie adresowe MA0—MA10. Ponieważ informacja na wyjściu pamięci obrazowej może się jeszcze zmienić, to dopiero narastające zbocze sygnału S-L/ powoduje zatrzaśnięcie danych. Ten sam sygnał wyzwala rejestr przesuwny 74165, co powoduje wysłanie kolejnych bitów poprzedniego znaku do miksera video. Przesyłanie danych z pamięci obrazowej do generatora znaków, a następnie do rejestru przesuwnego, powoduje opóźnienie o dwa cykle zegara znaków (od momentu zaadresowania pamięci obrazowej do momentu, gdy dana opuszcza rejestr przesuwny).

W obwodzie miksera video sygnały VIDEO, CURSOR, DISPEN, S-L/, HSYNC i VSYNC łączą się tworząc zespolony sygnał wizyjny.

Sygnały CURSOR i DISPEN z układu CRTC są przy użyciu dwóch układów 74174 opóźniane o dwa cykle zegara znaków, tak aby skompensować zwłokę dwóch znaków, o której była mowa wcześniej. Układ 74174 jest taktowany odwróconym zegarem znaków (CC/), ponieważ sygnały na jego wyjściach są zmieniane przez narastające zbocze zegara, odwrotnie niż w CRTC, którego wyjścia zmieniają się przy opadającym zboczu. Opóźnione sygnały CURSOR i DISPEN wraz z sygnałem S-L/, zmieszane z sygnałem VIDEO, tworzą sygnał bezpośrednio sterujący jasnością plamki.

Przesunięcia między wspomnianymi sygnałami a sygnałem VIDEO powodują, że sumaryczny sygnał jest nieco „zaśmiecony". Sterowany sygnałem DOT przerzutnik typu D wydziela prawidłowy sygnał jaskrawości. Jest on następnie mieszany przez tranzystor BD 140 z sygnałami HSYNC i VSYNC, co w sumie daje zespolony sygnał wizyjny. Tranzystor BD 139 pracujący w układzie wtórnika emiterowego dostosowuje ten sygnał do 75-omowego wejścia monitora.

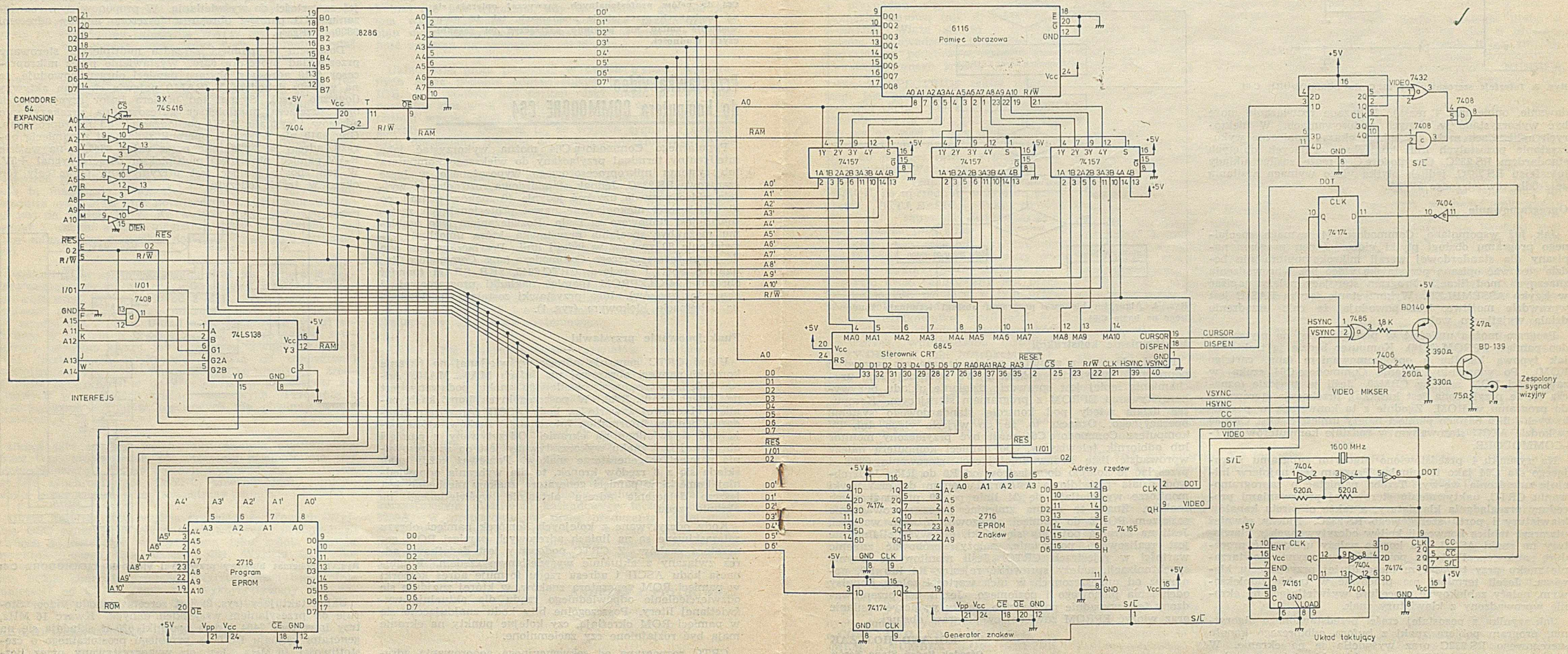
W bloku sterowania dostępem do pamięci trzy układy 74157 (czterowejściowe selektory linii danych) tworzą multiplekser adresów. Linie A11 ... A15 z mikroprocesora podawane są na układ 74LS138 (dekoder 1 z 8), który generuje sygnał RAM/. Gdy mikroprocesor generuje dostęp do pamięci obrazu, sygnał RAM/ przechodzi w stan 0, co powoduje wybór wejść A układów 74157. Gdy RAM/ przechodzi w stan 1, mikroprocesor nie korzysta z pamięci obrazu i wybierane są wejścia B przekazujące sygnały z linii adresowych MA0 ... MA10 do pamięci obrazu. Sygnał RAM/ doprowadzony jest również do układu 8286, co powoduje przyłączenie szyny danych do pamięci RAM wtedy, gdy mikroprocesor realizuje dostęp do pamięci obrazowej.

Jeżeli pamięć obrazu nie jest adresowana przez mikroprocesor, to bufory przechodzą w stan trzeci, izolując w ten sposób szynę 6510 od operacji realizowanych przez CRTC.

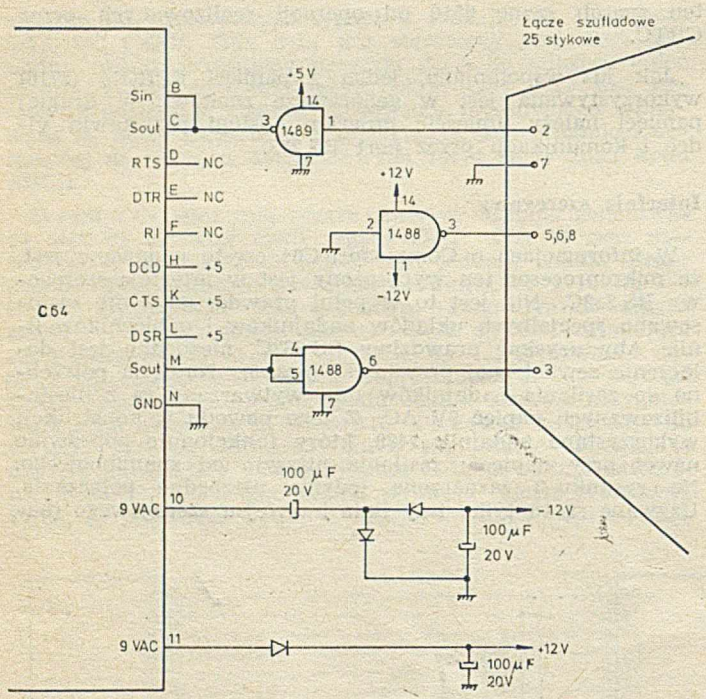
Jak już wspomniano, jedna z pamięci EPROM (2716) wykorzystywana jest w generatorze znaków. W drugiej pamięci należy umieścić program obsługi przystawki video i komunikacji przez port RS 232.

Interfejs szeregowy

W informacjach o Commodore C64 często podawane jest, że mikroprocesor ten wyposażony jest w interfejs szeregowy RS 232C. Nie jest to w pełni prawda, gdyż nie zastosowano specjalnych układów nadajników i odbiorników linii. Aby uzyskać prawdziwy RS 232C, niezbędne jest dołączenie zewnętrznej przystawki (rys. 3). Napięcia potrzebne do zasilania nadajników linii wytwarzane są z niestabilizowanych napięć 9V AC. Z tego powodu w konstrukcji wykorzystano nadajnik 1488, który funkcjonuje poprawnie nawet przy napięciu zasilania niższym od znamionowego. Na rysunku 3 zaznaczono jedynie niezbędne połączenia. Używane są tu tylko trzy linie interfejsu szeregowego (na-



Rys. 2. Schemat ideowy przystawki video do C 64, umożliwiającej wyświetlanie 80 znaków w wierszu



Rys. 3. Interfejs szeregowy RS 232 C do COMMODORE C 64

dawanie, odbieranie i masa). Pokazane rozwiązanie może być wykorzystane w różnych zastosowaniach. W niektórych aplikacjach może się jednak okazać konieczne wykorzystanie pozostałych linii kontrolno-sterujących objętych standardem RS 232C. Commodore C64 może zasilać układy interfejsu RS 232C, jednak płytka video wymaga zasilania ze źródła zewnętrznego 5V/1A.

Oprogramowanie

Jak już wspomniano, Commodore C64 wymaga specjalnego programu obsługi płytki video. Żaden program napisany dla standardowej wersji mikrokomputera nie będzie sterować opisaną przystawką video bez wprowadzenia znacznych modyfikacji. Program sterujący należy napisać w języku ASSEMBLER. Wykorzystanie języka BASIC jest wprawdzie możliwe, lecz powoduje, że całe urządzenie działa wyjątkowo wolno.

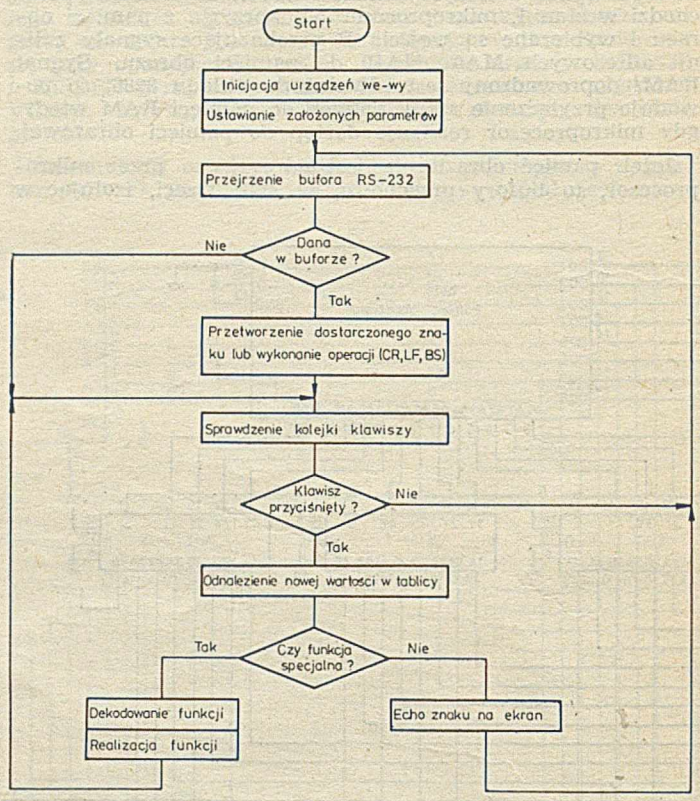
Program można umieścić w przewidzianej na płytce video pamięci EPROM (2716). Komputer będzie ją traktował jak typową kasetkę z oprogramowaniem (ang. cartrige ROM). Po włączeniu zasilania Commodore C64 szuka w zawartości ROM znaków CBM80. Jeśli je znajdzie (oznacza to, że przyłączona jest kasetka), przekazuje sterowanie do programu w ROM. Zgodnie z tą konwencją w początkowych kilku bajtach pamięci należy umieścić tekst CBM80 w kodzie ASCII stosowanym w rodzinie komputerów PET-COMMODORE.

Na rysunku 4 przedstawiono algorytm programu sterującego dla C64 jako terminala. Program ten najpierw inicjuje urządzenia we-wy. Trzeba pamiętać o zaprogramowaniu CRTC, uaktywnieniu sterowanej przerwaniami procedury przeglądania klawiatury, a także otwarciu kanałów klawiatury i portu szeregowego RS 232C. Należy również utworzyć tablicę konwersji kodów odczytywanych z klawiatury na znaki ASCII. Na koniec należy uaktywnić procedurę „auto-repeat", która powoduje samoczynne powtarzanie znaku przy trwającym nieco dłużej przyciśnięciu klawisza. Jeżeli terminal ma pracować w trybie dupleksowym, należy zablokować procedurę wyświetlającą na ekranie wprowadzony z klawiatury znak.

Jak wynika z pozostałej części przedstawionego algorytmu, program pobiera znaki z bufora odbiorczego kanału szeregowego RS 232C oraz wyświetla je na ekranie. W tym samym czasie odbiera on znaki z klawiatury, przekodowuje na ASCII i wysyła do bufora nadawczego kanału szeregowego. Jeśli terminal pracuje w trybie półdupleksowym (włączone jest lokalne echo), to dane z klawiatury są wysyłane także do pamięci obrazu.

W C64 przewidziano osiem specjalnych klawiszy funkcyjnych. Można je wykorzystać do realizacji funkcji związanych z obsługą terminala, np.:

f1: cofnięcie (ang. backspace)
f2: włączenie/wyłączenie lokalnego echa, czyli dupleks lub półdupleks
f3: przesłanie BREAK
f4: przejście na zapis wyłącznie dużych liter
f5: wymazanie zawartości ekranu
f6: powrót do zapisu normalnego (małe i duże litery)



Rys. 4. Algorytm blokowy programu obsługi terminala zbudowanego na bazie C 64

Uruchamianie konstrukcji

Przed włączeniem zasilania należy jeszcze raz sprawdzić poprawność wszystkich połączeń. We wstępnej fazie uruchamiania lepiej jest sprawdzić działanie układu bez wykorzystywania EPROM z programem sterującym. Komputer działa wtedy pod kontrolą standardowego systemu operacyjnego. Oznacza to, że do wyjścia video lub TV komputera Commodore C64 musi być przyłączony monitor lub odbiornik telewizyjny. Do pamięci komputera należy wprowadzić listę kodów inicjujących sterowanie CRT i przesyłać je kolejno do rejestrów od R0 do R15. Jeśli obwód działa prawidłowo, na przyłączonym do przystawki monitorze wyświetlają się 24 linie po 80 przypadkowych znaków. Znaki te można zmieniać przez wprowadzanie rozkazem POKE do pamięci obrazowej różnych wartości. Jeśli na ekranie pojawiły się znaki, ale wyświetlanie nie jest realizowane poprawnie, należy spróbować zmienić wartości w rejestrach CRTC. Jeśli monitor nie trzyma synchronizacji, należy sprawdzić rejestry R4 lub R0, ponieważ od umieszczonych w nich wartości zależą impulsy odchylania pionowego i poziomego. Jeżeli tak przeprowadzony test wypadnie pozytywnie, należy wyłączyć zasilanie oraz włożyć EPROM zawierający program obsługi.

ROLAND HOJCZAK
Ośrodek Badań Stanu Torów
Warszawa

Można by zapełnić rzekę deficytowego w naszym kraju papieru dyskutując nad rozlicznymi ograniczeniami, jakim musi stawić czoło użytkownik ZX SPECTRUM. W porównaniu z wprowadzanymi obecnie na rynek nowymi typami komputerów, dotkliwą bolączką dla stosunkowo szerokiej rzeszy użytkowników ZX Spectrum jest brak w nim wielu użytecznych instrukcji. INFORMATYKA publikowała już opis rozszerzonej normy języka BASIC — ANSI, w świetle której jest to niemal zupełnie inny (jakościowo) język.

Opisywany niżej program daleki jest od tak szeroko potraktowanych zmian. Tym niemniej, zestawiając obszar pamięci, który traci użytkownik, z dodatkowymi możliwościami, które mu daje, należy ocenić program BETA BASIC pozytywnie.

Program BETA BASIC

Dla większości mikrokomputerów oferowane są programy rozszerzające przewidziany przez producenta zestaw instrukcji języka BASIC. Dodatkowe instrukcje znacznie ułatwiają pracę na komputerze. Zalety takiego rozwiązania są szczególnie odczuwalne w przypadku współpracy z szybką pamięcią masową (np. dyskami elastycznymi). Wiąże się to z krótkim czasem wprowadzania programu rozszerzającego podstawowy zasób słów wykorzystywanego języka.

Dla ZX SPECTRUM takim programem jest program BETA BASIC firmy BETASOFT. W artykule opisana zostanie wersja 1.8 z 1984 r. Program ten ma ok. 9000 bajtów długości. Jest to istotne, gdyż musi on zostać wczytany do pamięci razem z programem użytkownika. Wczytanie programu użytkownika realizujemy instrukcją **MERGE**, po uprzednim wczytaniu programu BETA BASIC. Taki sposób wczytania zapewnia automatyczny start naszego programu.

Istnieją dwa sposoby wywoływania dodatkowych instrukcji. Pierwszy polega na wykorzystaniu znaków graficznych; drugi realizowany jest przez instrukcję **FN**, pełniącą w tym przypadku rolę pośrednika. Aby nie pozbawiać użytkownika możliwości korzystania ze standardowych znaków graficznych, wprowadzono instrukcję **KEYWORDS**. I tak, instrukcja **KEYWORDS 1** przyporządkowuje klawiszom nowe słowa, a **KEYWORDS 0** standardowe znaki graficzne (wszystko z kursorem G).

Łącznie poprzez znaki graficzne można odwołać się do 35 dodatkowych instrukcji:

A-**ALTER**, C-**CLOCK**, D-**DO**, E-**ELSE**, F-**FILL**, G-**GET**, I-**EXIT IF**, J-**WHILE**, K-**UNTIL**, L-**LOOP**, M-**SORT**, N-**ON ERROR**, O-**ON**, P-**DPOKE**, R-**ROLL**, S-**SCROLL**, T-**TRACE**, U-**USING**, V-**RND**, W- **INKEY$**, Y-**PI**, Y-**FN**, Z-**POINT**, 0-**POP**, 1-**DEF PROC**, 2-**PROC**, 3-**END PROC**, 4-**RENUM**, 6-**AUTO**, 7- **DELETE**, 8- **KEYWORDS**, !-**DEF KEY**, $-**KEY IN**, %-**EDIT** (dla trybu graficznego), &-**JOIN**, 0-**EDIT** (dla trybu niegraficznego).

Przez FN wywołujemy 21 kolejnych, dodatkowych instrukcji: a-**AND**, c-**COSE**, f-**FILLED**, i-**INSTRING**, m-**MEM**, n-**NUMBER**, o-**OR**, p-**DPEEK**, r-**RNDM**, s-**SINE**, v-**MOD**, x-**XOR**, b$-**BIN$**, c$-**CHAR$**, h$-**HEX$**, k$-**SCRN$**, m$-**MEMOR$**, s$-**STRING$**, t$-**TIME$**, u$-**USING$**. Dużym ułatwieniem jest fakt, że zawsze możemy odwołać się (przez ENTER) do strony, na której podany jest opis argumentów instrukcji wywoływanych przez FN. Dla przykładu, zapis **FN m$ / /** oznacza kolejność wprowadzania z klawiatury, a zarazem informuje, że instrukcja jest bezargumentowa. Nazwa instrukcji, w tym przypadku **MEMOR$**, pojawi się na ekranie po wciśnięciu klawisza „(" (otwarcie nawiasu).

Ogólnie biorąc, nowe instrukcje można pogrupować następująco:

- poprawiające komfort pracy z komputerem
- upraszczające strukturę programu
- pozwalające lepiej wykorzystać pamięć komputera
- stwarzające nowe możliwości graficzne.

Jedna z ciekawszych instrukcji jest **CLOCK**. Ma ona 8 trybów, dających następujące kombinacje: wskazywanie czasu, skok do zdefiniowanego przez użytkownika podprogramu obsługi i włączenie alarmu dźwiękowego. Tryb 7 zapewnia jednoczesną realizację wszystkich trzech możliwości. Normalnie zegar startuje z czasem zerowym, licząc od chwili włączenia zasilania. Odmiana tej instrukcji jest **TIME$**. Po wywołaniu tej instrukcji wskazywany jest czas momentu wywołania.

AUTO ⟨start⟩/⟨start, krok⟩ uwalnia użytkownika od kłopotliwego wprowadzania kolejnego numeru linii programu. Jeżeli nie podamy parametru krok, to standardowo jest on ustawiony na 10.

DELETE (nr wiersza) TO (nr wiersza) usuwa wybrany fragment programu.

Instrukcja warunkowa „jeśli" ma postać **IF** ⟨wyrażenie logiczne⟩ **THEN** ⟨instrukcja⟩ /:**ELSE** ⟨instrukcja⟩⟩. Cecha charakterystyczna jest konieczność umieszczenia dwukropka przed ELSE.

W BETA BASIC wprowadzono na wzór języka PASCAL, instrukcje iteracyjne tworzone za pomocą słów: **DO, WHILE, UNTIL, LOOP, EXIT IF**. Również z PASCALA zaczerpnięte są trzy instrukcje do tworzenia i wywoływania procedur: **DEF PROC, PROC, END PROC**. Konstrukcja wyliczanego skoku oraz wyliczanego wywołania podprogramu jest odmienna od typowej i ma postać: **GOTO ON, GOSUB ON**.

ON ERROR (nr linii) pozwala na obsługę błędu, przez skok do podprogramu zaczynającego się od podanego jako parametr numeru linii.

POP (zmienna) usuwa ze stosu adres. Dzięki temu można wychodzić z procedur i pętli, nie robiąc zamieszania w stosie. Metodę taką stosuje się w podprogramach do powrotu do programu głównego bez używania RETURN. Czasami jest to bardzo wygodne.

SORT porządkuje tablicę numeryczną lub łańcuch od wartości najmniejszej do największej (ewentualnie wykorzystując kody ASCII).

SORT INVERSE sortuje odwrotnie. Przykładowa konstrukcja: **SORT a$ (1 TO 7)**.

CHAR$ (liczba) zamienia liczbę z przedziału 1—2^{16} na ciąg o długości d w ó c h liter. Pozwala to na zmniejszenie obszaru pamięci zajmowanego przez tablicę numeryczną. **NUMBER** (dwie litery) działa w kierunku odwrotnym.

DEC zamienia liczby z postaci szesnastkowej na dziesiętną.

HEX$ działa w przeciwnym kierunku.

DPEEK wprowadza zawartość dwóch kolejnych komórek pamięci. Przyjęta jest tu konwencja, że pod podanym jako parametr adresem umieszczony jest mniej znaczący bajt, a pod adresem o 1 większym — bardziej znaczący. Oba bajty łączone są w liczbę 16-bitową.

DPOKE ustawia zawartość dwóch kolejnych komórek pamięci według podanej jako parametr liczby 16-bitowej (0 ... 65535).

MEM / / podaje jako wynik liczbę wolnych bajtów pamięci. Jest odpowiednikiem sekwencji PRINT 65535 — — USR 7962.

INSTRING (start, a$, b$) szuka w łańcuchu a$ łańcucha b$ zaczynając od elementu łańcucha a$ start.

**STRING$** (n, a$) powtarza łańcuch a$ zadany przez parametr n liczbę razy.

USING jest instrukcją formatującą wyprowadzaną informację. Przykładowa konstrukcja PRINT USING ("## · ·##" : wyrażenie).

ROLL kierunek :⟨x, y : szer, wys⟩ umożliwia przesuwanie wycinka ekranu (okna) w podstawowych kierunkach o jeden punkt. Kierunek określany jest jako liczba 5 ... 8 według zamieszczonych na klawiaturze strzałek: x i y jak w PLOT; szer = 1 ... 32, wys = 1 ... 176.

SCROLL przesuwa obraz na ekranie o wiersz do góry. SCROLL kierunek — przemieszcza obraz na ekranie o jeden punkt w zadanym kierunku. To, co wychodzi poza ekran, jest kasowane.

Oprócz wprowadzenia dodatkowych instrukcji w BETA BASIC rozszerzono również możliwości niektórych instrukcji dotychczasowych.

PLOT x, y ;⟨łańcuch⟩ wpisuje w dowolnym miejscu ekranu, z dokładnością do jednego punktu, łańcuch o długości do 32 znaków. Warunkiem koniecznym dla funkcjonowania instrukcji jest wciśnięcie **PLOT** przy pracy w trybie niegraficznym (KEYWORDS 0). Możliwości PLOT i DRAW dodatkowo poszerzają zmienne skalowania **XOS**, **YOS**, **XRG**, **YRG**. Wartości początkowe wynoszą odpowiednio 0, 0, 256, 176. Wartości tych zmiennych nie są zmieniane przez RUN i CLEAR. XOS i YOS powodują przesunięcie początku układu współrzędnych, natomiast XRG i YRG — przesunięcie wzdłuż osi x i y.

Nawet tak pobieżny przegląd możliwości programu BETA BASIC powinien przekonać Czytelników o jego dużej użyteczności. Przy tworzeniu nieco dłuższych programów w pełni rekompensuje się 9 KB pamięci zajętych przez BETA BASIC. Wynika to z możliwości zastąpienia wielu instrukcji języka ZX SPECTRUM BASIC czasem nawet pojedynczą instrukcją BETA BASIC.

JERZY KRYSTEK
ADAM MIELCZAREK
Zabrze

Tematyka grafiki mikrokomputerowej przyciąga coraz więcej uwagi. Planowaliśmy poświęcić jej więcej uwagi niż nam się w 1985 r. udało. Dlatego też z pewnością poświęcimy jej jeszcze sporo miejsca w przyszłości, choć zapewne formuła wysokonakładowego pisma, jakim ma się stać mikroKLAN spowoduje, że będziemy musieli opracować bardziej przystępną dla szerszego kręgu czytelników formę. Będziemy jednak zapewne mogli, pisząc o grafice, również ją pokazać.

Antoni Urban w kolejnym materiale związanym z normą jądra graficznego GKS pisze o rysowaniu różnych łuków — w tym okręgów i paraboli. Jest to zagadnienie, z którym musimy sobie jakoś poradzić niemal przy każdej grafice. Efektywny algorytm (tu akurat liczy się szybkość działania i dokładność) to główny kierunek poszukiwań programistów wykorzystujących mikrokomputery.

Łuki krzywych gładkich

Rysowanie łuków krzywych gładkich nie jest objęte normą GKS. Wśród prymitywów nie ma reprezentacji graficznej łuków okręgu, elipsy, hiperboli, paraboli, nie mówiąc już o bardziej skomplikowanych krzywych. Norma GKS wymienia jedynie prymityw o nazwie Generalized Drawing Primitive (w skrócie G.D.P.), pod którym autorzy implementacji mogą przewidzieć krzywe, które ich zdaniem będą wykorzystywane przez użytkowników jądra graficznego. W przypadku realizowania implementacji normy GKS dla sprzętu mikrokomputerowego ważny jest jeszcze jeden czynnik: możliwości obliczeniowe sprzętu. Czynnik ten nie tylko decyduje o wyborze takich a nie innych krzywych, lecz również zmusza do szukania najbardziej optymalnych algorytmów interpolacji. I to właśnie, skromnym zdaniem autora, jest bakcylem mikroinformatyki, który raz połknięty zaraża na całe życie.

Rozpatrzmy następujący przykład: jak narysować okrąg, gdy brak jest interpolatora kołowego? Problem rozwiązać można na wiele sposobów.

1. Okrąg o środku **S(a, b)** i promieniu **r** zastępowany jest wielokątem o **n** wierzchołkach, których współrzędne wyrażają się wzorami:

$$x_i = a + r \cdot \cos(i \cdot \varphi)$$
$$y_i = b + r \cdot \sin(i \cdot \varphi)$$

gdzie **i** przyjmuje kolejne wartości od **0** do **n—1**. W opisie nie uwzględniono kątów początkowego i końcowego łuku.

2. Idea rozwiązania jest identyczna jak w poprzednim rozwiązaniu, przy czym współrzędne kolejnych wierzchołków wielokąta obliczane są w sposób rekurencyjny według wzorów:

$$x' = (x-a)\cdot\cos\varphi - (y-b)\cdot\sin\varphi + a$$
$$y' = (y-b)\cdot\cos\varphi + (x-a)\cdot\sin\varphi + b$$

Ponieważ w tym przypadku kąt φ jest dostatecznie mały, można zamiast funkcji trygonometrycznych zastosować przybliżenie wynikające z rozwinięcia szeregu Fouriera:

$$\sin\varphi = \varphi - \frac{\varphi^3}{6} + \dots$$

$$\cos\varphi = 1 - \frac{\varphi^2}{2} + \dots$$

3. Kreślone są jednostkowe odcinki między sąsiednimi punktami w takim kierunku, by zachować minimalną odległość od środka okręgu. Rozwiązanie to jest podobne do rozwiązania następnego, ale wzór, według którego wybierany jest optymalny kierunek, nie gwarantuje zadowalającej szybkości.

$$d^2 = (x-a)^2 + (y-b)^2$$

Żadne z dotąd omówionych rozwiązań nie daje zadowalających rezultatów; w pierwszym konieczne jest wyliczanie wartości funkcji sinus i cosinus, wszystkie odwołują się do mnożenia.

Okazuje się, że można narysować okrąg wykonując tylko dodawanie, odejmowanie i porównywanie liczb całkowitych. Czytelnik łatwo stwierdzi, że jest to rozwiązanie najlepsze, zapewniające największą dokładność interpolacji, dużą szybkość rysowania, a co najważniejsze — nie powodujące występowania błędów przy źle dobranych parametrach wejściowych. Procedura wymaga zadeklarowania współrzędnych trzech punktów na płaszczyźnie (środek okręgu S, początek łuku A, koniec łuku B) oraz kierunku poruszania się po okręgu — zgodnie z ruchem wskazówek zegara (k = —1) lub przeciwnie do ruchu wskazówek zegara (k = +1). Ze zrozumiałych względów końce łuku nie mogą być równe środkowi okręgu. Może być natomiast tak, że początek i koniec łuku są tym samym punktem — rysowany jest wtedy pełny okrąg. Algorytm interpolacji kołowej w niezawodny sposób bada warunek zakończenia rysowania, nawet wtedy, gdy punkt końcowy zadeklarowany został w innej odległości od środka niż początek łuku. Interpolacja kończy się w momencie przecięcia półprostej $SB^{\rightarrow}$. Algorytm wykorzystuje ten sam aparat matematyczny, który wymieniono przy demonstrowaniu interpolatora liniowego (zob. „Wielolinia", mikroKLAN 10'85).

Aby lepiej zrozumieć ten algorytm, należy pamiętać, że funkcja

$$(x, y) = (x-a)^2 + (y-b)^2 - r^2$$

przedstawia okrąg, a funkcja:

$$g(x, y) = \begin{vmatrix} x-a & y-b \\ x_1-a & y_1-b \end{vmatrix} \cdot k$$

przedstawia prostą łączącą środek okręgu i koniec łuku. W tym ostatnim wzorze uwzględniono kierunek interpolacji z tego względu, by zmiana znaku wartości przyjmowanej przez funkcje g z „+" na „—" oznaczała przecięcie półprostej (a nie prostej) $SB^{\rightarrow}$. Na rysunku 1 przedstawiono graf algorytmu.

Po zapoznaniu się z procedurą interpolacji kołowej należy powrócić do języka normy GKS, by zobaczyć składnię instrukcji rysowania łuków okręgu.

GDP (n, p)

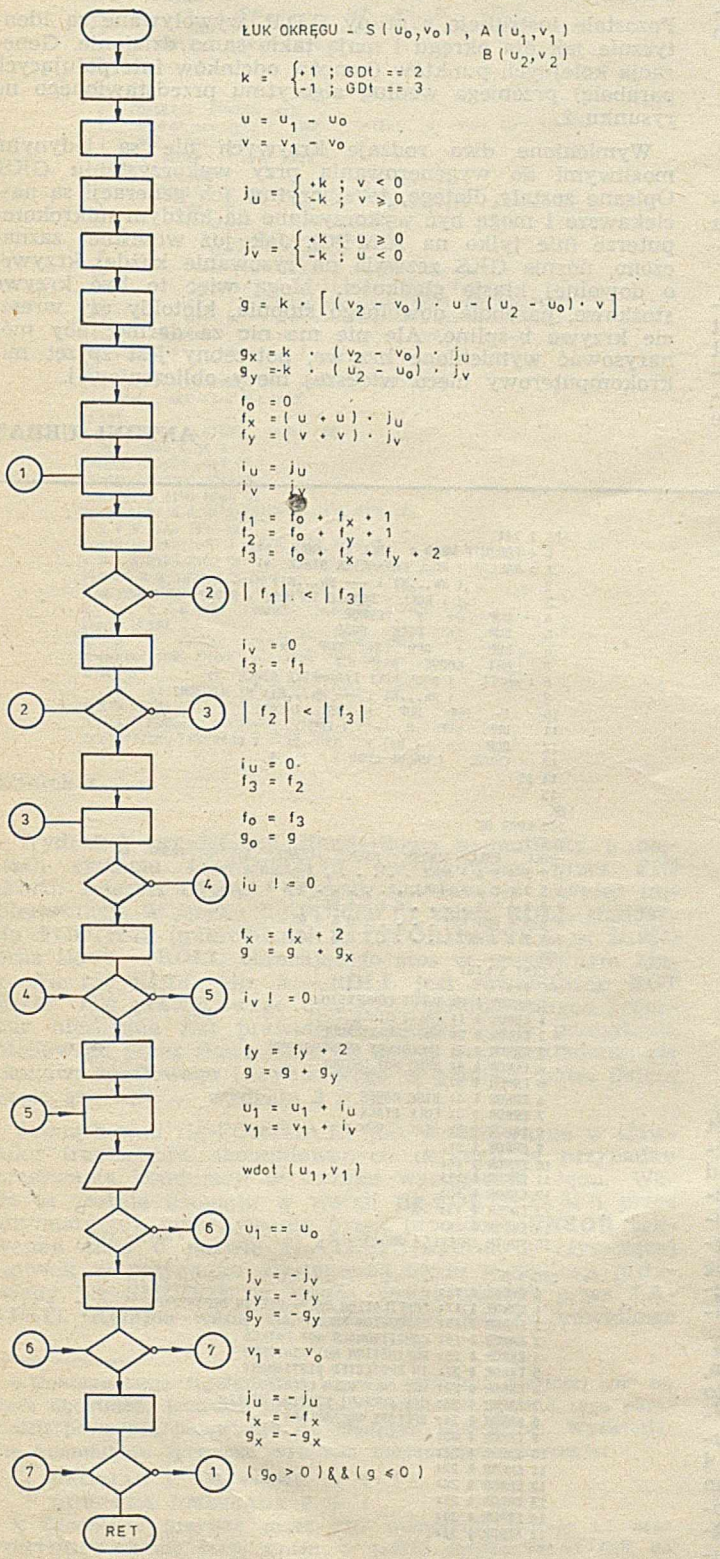
gdzie: **int n** — liczba punktów, **float p [ ] [2]** — współrzędne punktów **A, S, B**

Pojedynczą instrukcją GDP można narysować kilka łuków okręgu pod warunkiem, że wszystkie rysowane są w tym samym kierunku.

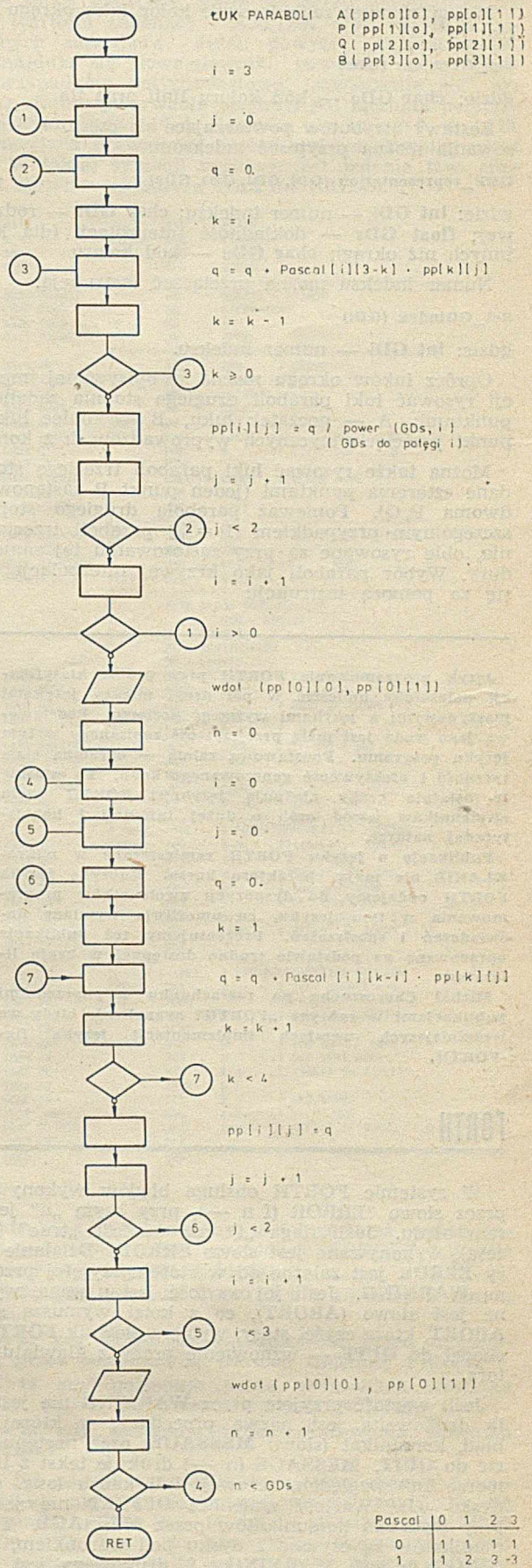
Kierunek interpolacji zadaje się instrukcją:

Set_GDtype (GDt)

gdzie: **char GDt** — rodzaj krzywej, **GDt = 2** — łuk okręgu rysowany zgodnie z ruchem wskazówek zegara (CW), **GDT = 3** — łuk okręgu rysowany przeciwnie do ruchu wskazówek zegara (CCW).



Rys. 1



Pascal	0	1	2	3
0	1	1	1	1
1	1	2	3	-3
2	1	3	-6	3
3	1	-3	3	-1

Rys. 2

Można również zadeklarować kolor łuku okręgu instrukcją:

Set_GDcolour (GDc)

gdzie: **char GDc** — kod koloru linii oraz tła

Zestawy atrybutów powtarzające się często w trakcie rysowania można przypisać indeksom:

GDP_representation (GDi, GDt, GDs, GDc)

gdzie: **int GDi** — numer indeksu; **char GDt** — rodzaj krzywej; **float GDs** — dokładność interpolacji (dla krzywych innych niż okrąg); **char GDc** — kod koloru

Numer indeksu można przełączać instrukcją:

Set_GDindex (GDi)

gdzie: **int GDi** — numer indeksu.

Oprócz łuków okręgu można w opisywanej implementacji rysować łuki paraboli drugiego stopnia zadane trzema punktami: A — początek łuku, B — koniec łuku, P — punkt przecięcia stycznych wyprowadzonych z końców.

Można także rysować łuki paraboli trzeciego stopnia zadane czterema punktami (jeden punkt P zastępowany jest dwoma P, Q). Ponieważ parabola drugiego stopnia jest szczególnym przypadkiem (P = Q) paraboli trzeciego stopnia, obie rysowane są przy zastosowaniu tej samej procedury. Wybór paraboli jako krzywej interpolacji dokonuje się za pomocą instrukcji:

Set_GDtype (GDt)

gdzie: **char GDt** — rodzaj krzywej, **GDt = 0** — parabola rysowana punktami, **GDt = 1** — parabola rysowana odcinkami.

Atrybut GDsize oznacza w tym przypadku ilość punktów przybliżających parabolę, a więc dokładność interpolacji.

Set_GDsize (GDs)

gdzie: **float GDs** — dokładność interpolacji.

Pozostałe instrukcje z grupy G.D.P. wywoływane są identycznie jak dla okręgu i mają takie samo działanie. Generacja kolejnych punktów (końców odcinków interpolujących parabolę) przebiega według algorytmu przedstawionego na rysunku 2.

Wymienione dwa rodzaje krzywych nie są jedynymi możliwymi do wygenerowania przy wykorzystaniu GKS. Opisane zostały dlatego, że algorytmy ich generacji są najciekawsze i mogą być wykorzystane na każdym mikrokomputerze (nie tylko na IBM PC). Jak już wcześniej zaznaczono, norma GKS zezwala na rysowanie każdej krzywej o dowolnej klasie gładkości. Mogą więc to być krzywe stożkowe, parabole dowolnego stopnia, klotoidy czy wreszcie krzywe b-spline. Ale nie ma nic za darmo; aby móc narysować wymienione krzywe, potrzebny jest sprzęt mikrokomputerowy nieco większej mocy obliczeniowej.

ANTONI URBAN

Język programowania FORTH przy próbie klasyfikacji należałoby umieścić w pół drogi między językami maszynowymi a językami wyższego poziomu. Podstawową jego wadą jest mała przejrzystość zapisanego w tym języku programu. Podstawową zaletą — ogromna elastyczność i efektywność generowanego kodu. To właśnie te ostatnie cechy zjednują językowi FORTH wielu zwolenników wśród osób o dużej inwencji i hobbistycznej naturze.

Publikacje o języku FORTH zamieszczane w mikroKLANIE nie mają charakteru kursu. Rubrykę języka FORTH oddajemy do dyspozycji zwolenników programowania w tym języku, co umożliwia wymianę doświadczeń i spostrzeżeń. Prezentujemy też publikacje opracowane na podstawie trudno dostępnej w kraju literatury.

Michał Choroszucha po rozrachunku z poprzednimi publikacjami w rubryce „FORTH" sygnalizuje błędy we wcześniejszych wersjach implementacji języka fig-FORTH.

FORTH

W systemie FORTH obsługa błędów wykonywana jest przez słowo ?ERROR (f n ---), przy czym „n" jest numerem błędu. Jeśli flaga „f" ma wartość „true" (różną od zera), wykonywane jest słowo **ERROR**. Działanie procedury **ERROR** jest zależne od wartości przyjętej przez zmienną **WARNING**. Jeśli jej wartość jest ujemna, wykonywane jest słowo (**ABORT**), co z kolei wymusza sekwencję **ABORT**, która czyści stosy, wybiera słownik **FORTH** i przechodzi do **QUIT** — wznowienie pracy z klawiatury operatora.

Jeśli wartość przyjęta przez **WARNING** nie jest ujemna, to drukowana jest nazwa procedury, w której wykryto błąd, komunikat (słowo **MESSAGE**) oraz następuje przejście do **QUIT**. **MESSAGE** (n ---) drukuje tekst z linii o numerze „n" względem zerowej linii kadru (ang. screen) 4 dysku 0 [1]). Wartość zmiennej **OFFSET** nie jest istotna przy wydruku komunikatów przez **MESSAGE**. Teksty komunikatów są czytane z dysku pod warunkiem, że **WARNING** > 0; jeśli **WARNING** = 0, drukowany jest tylko numer komunikatu.

```
SRC # 241
  0 ( FORTH79 WORD :  ROLL ; -ROLL  **)
  1 : ROLL    ( ROLL PARAMETER STACK  *)
  2           ( Nk...N1 i --- Nk...Ni+ Ni-...N1 Ni )
  3           ( i ROLL = ROTATE TOPMOST i DIGITS ON STACK )
  4     DUP   0=   9   ?ERROR
  5     DUP   1+   PICK   SWAP
  6     DUP   +   SP@   2+   DUP   2+
  7     ROT   CMOVE   DROP   ;
  8 : -ROLL     ( BACK ROLL PARAMETER STACK  *)
  9           ( Nk...N1 i --- Nk...Ni+ N1 Ni...N2 )
 10     1-   DUP   DUP   +   )R   (HEAP)
 11     DUP   DUP   R   -   ( SRC )
 12     DUP   2-   ( DST )   R)   2+   ( QUAN )
 13     CMOVE   ( VALUE ADDR )   !   ;
 14 ;S
 15
OK
240 LOAD OK
VLIST
-ROLL   ROLL   !HEAP   PICK   (HEAP)   DEPTH   TASK   OK
1 2 3 4 5 6 4 ROLL . . . . . 3 6 5 4 2 1 OK
1 2 3 4 5 6 4 -ROLL . . . . . 5 4 3 6 2 1 OK
9 8 7 6 ROT . . . . 8 6 7 9 OK
9 8 7 6 ROT ROT . . . . 7 8 6 9 OK
9 8 7 6 3 -ROLL . . . . 7 8 6 9 OK
 OK
4 LIST 5 LIST
SRC # 4
  0 ERROR # 0: INPUT CONVERSION
  1 ERROR # 1: EMPTY STACK
  2 ERROR # 2: DICTIONARY FULL
  3 ERROR # 3: INCORRECT ADDRESS MODE
  4 ERROR # 4: NAME NOT UNIQUE
  5 ERROR # 5:
  6 ERROR # 6: DISC RANGE ?
  7 ERROR # 7: FULL STACK
  8 ERROR # 8: DISC ERROR !
  9 ERROR # 9:
 10 ERROR # 10:
 11 ERROR # 11:
 12 ERROR # 12:
 13 ERROR # 13:
 14 ERROR # 14:
 15 FORTH INTEREST GROUP                          MAY 1, 1979

SRC # 5
  0 ERROR # 16:
  1 ERROR # 17: COMPILATION ONLY, USE IN DEFINITION
  2 ERROR # 18: EXECUTION ONLY
  3 ERROR # 19: CONDITIONALS NOT PAIRED
  4 ERROR # 20: DEFINITION NOT FINISHED
  5 ERROR # 21: IN PROTECTED DICTIONARY
  6 ERROR # 22: USE ONLY WHEN LOADING
  7 ERROR # 23: OFF CURRENT EDITING SCREEN
  8 ERROR # 24: DECLARE VOCABULARY
  9 ERROR # 25:
 10 ERROR # 26:
 11 ERROR # 27:
 12 ERROR # 28:
 13 ERROR # 29:
 14 ERROR # 30:
 15 ERROR # 31:
OK
```

Wydruk 1

```
CR 240 LIST 241 LIST

SRC # 240
  0 ( ?LATEST   FORGET  **)
  1 : ?LATEST   ( ---   Verify Dictionary Pointer  *)
  2    LATEST   C@   [ HEX ] 20   AND
  3       IF   ( Bad entry - Compilation error )
  4          LATEST   DUP   DP   !
  5          PFA   LFA   @   CURRENT   @   !
  6       ENDIF   ;
  7 : FORGET   ( ---   FORGET name  *)
  8    CURRENT   @   CONTEXT   @   -   [ HEX ]   18   ?ERROR
  9    [COMPILE]   '   DUP   FENCE   @   U<   15   ?ERROR
 10    ( PFA'   Following modified )   NFA   VOC-LINK
 11       BEGIN
 12          @   ( NFA' pvla )   DUP   0=   0E   ?ERROR
 13          2DUP   U<   ( NFA' ( pvla ? )   0=
 14       UNTIL   ( NFA' > pvla )
 15 --)

SRC # 241
  0 ( FORGET -   CONT'D  **)
  1    ( NFA' new_pvla )   DUP   CONTEXT   @   U<   0D   ?ERROR
  2    DUP   VOC-LINK   !
  3       BEGIN   ( NFA' pvla )
  4          -DUP
  5       WHILE   ( Vocabulary found )
  6          )R   R   2-   )R   R   ( NFA' pla )   @
  7             BEGIN   ( NFA' nfa/pnfa )
  8                2DUP   SWAP   U<   0=
  9             WHILE   ( NFA' (= nfa )
 10                PFA   LFA   @   ( NFA' new_nfa )
 11             REPEAT   ( Until  nfa ( NFA'  found )
 12          R)   !   ( Store new vocab. limit NFA into PLA )
 13          R)   @   ( NFA' new_pvla   Process next vocabulary )
 14       REPEAT   ( Until end of vocab. link chain )
 15    ( NFA' )   DP   !   ;   ;S
OK
240 LOAD FORGET ERROR # 4: NAME NOT UNIQUE  OK
: & HERE . VOC-LINK @ . ; OK
VLIST
&   FORGET   ?LATEST   TASK   OK
& 2190 2064 : GG ; OK
SMUDGE & 2199 2064 OK
?LATEST & 2190 2064 OK
?LATEST & 2190 2064 OK
VOCABULARY A A DEFINITIONS & 219E 219C OK
: X ; & 21A6 219C OK
FORTH DEFINITIONS : C ; & 21AE 219C OK
: D ; & 21B6 219C OK
FORGET D & 21AE 219C OK
A VLIST
X      C   A   &   FORGET   ?LATEST   TASK   OK
FORTH VLIST
C   A   &   FORGET   ?LATEST   TASK   OK
A DEFINITIONS FORGET X & 219E 219C OK
A VLIST
    A   &   FORGET   ?LATEST   TASK   OK
FORTH VLIST
A   &   FORGET   ?LATEST   TASK   OK
DEFINITIONS FORGET A & 2190 2064 OK
BYE
```

Wydruk 2

Wydruk 1 przedstawia standardowe komunikaty o błędach systemu fig-FORTH 1.1 opublikowane przez FIG (Forth Interest Group). Wydruk ten zawiera również implementację w języku fig-FORTH 1.1 słowa **ROLL** standardu FORTH 79 (mikroKLAN 2, INFORMATYKA nr 2/1984) oraz słowa **—ROLL** obracającego stos w przeciwnym kierunku niż ROLL (np. 3 **—ROLL** jest równoważne **ROT ROT**). Dla prawidłowego działania opublikowanych procedur niezbędne jest prawidłowe wykonywanie przesyłania blokowego przez słowo **CMOVE** nawet przy nakładaniu się bloków: źródłowego i wynikowego w pamięci (patrz dalsza część artykułu — wydruki 3 i 4).

Jedną z wad fig-FORTHA 1.1 jest pozostawienie w słowniku fragmentów skompilowanych definicji w przypadku przerwania kompilacji w wyniku wystąpienia błędu. Wada ta została usunięta w wersji fig-FORTH 2.1 M [2]) przez automatyczne wykonywanie przez procedurę **ERROR** sekwencji słów o nazwie **?LATEST** (wydruk 2) czyszczącej słownik w przypadku wystąpienia błędu kompilacji (ustawiony bit SMUDGE w słowie wskazywanym przez **LATEST**). Kolejną wadą fig-FORTHA 1.1 jest nieprawidłowe działanie słowa **FORGET**, które w wersji oryginalnej nie zmienia wartości zmiennej VOC-LINK oraz pól PLA poszczególnych słowników. Jeżeli powyżej „zapomnianego" słowa znajdują się słowa-słowniki, powoduje to w większości przypadków natychmiastowe „zawieszenie się" systemu na skutek przerwania ciągłości pól łącznika poszczególnych definicji (od PLA słownika do link = 0 w LIT), co uniemożliwia prawidłowe przeszukiwanie słownika przez (FIND). W takiej sytuacji możliwy jest jedynie tzw. zimny start systemu od adresu HEX 100 (COLD).

```
0145 =          NEXT    EQU     0145H
0303 =          QTERM   EQU     0303H
16AA =          PCR     EQU     16AAH
030B            ORG     030BH
                ;---------------------------------------
030B 8243D2             DB      82H,'C','R'+80H
030E F702               DW      QTERM-0CH
0310 1203       CR      DW      $+2
0312 C32B03             JMP     XPCR
                ;---------------------------------------
0315 85                 DB      85H     ; CMOVE
0316 434D4F56           DB      'CMOV'
031A C5                 DB      'E'+80H
031B 0B03               DW      CR-5
031D 1F03       CMOVE   DW      $+2
031F 6960               MOV     L,C     !       MOV     H,B
0321 C1D1               POP     B       !       POP     D
0323 E3                 XTHL
0324 CD881A             CALL    CMOVX   ; BLOCK MOVE
0327 C1                 POP     B
0328 C34501             JMP     NEXT
                ;---------------------------------------
032B 210000     XPCR    LXI     H,0     ; CLEAR OUT VARIABLE
032E 22BA3B             SHLD    3BBAH
0331 C3AA1600           JMP     PCR     !       NOP
0335 =          XEND    EQU     $
                ;---------------------------------------
                ;       BYTE BLOCK MOVE ROUTINE
                ;               MCH - 850222
                ;       HL = SOURCE
                ;       DE = DESTINATION
                ;       BC = LENGTH
1A88                    ORG     1A88H   ; 1ST FREE CELL
1A88 78B1       CMOVX   MOV     A,B     !       ORA     C
1A8A C8                 RZ              ; QUIT IF #CHRS = 0
1A8B 7C         CMOV1   MOV     A,H
1A8C BA                 CMP     D       ; (H) - (D)
1A8D DA941A             JC      CMOV3   ; IF SRC < DST
1A90 C2991A             JNZ     CMOV2   ; IF >
1A93 7DBB               MOV     A,L     !       CMP     E
1A95 DA941A             JC      CMOV3   ; IF <
1A98 C8                 RZ              ; IF =
                ;---------------------------------------
                ; (HL) > (DE)   ; POST-INCREMENTING VERSION
1A99 7E         CMOV2   MOV     A,M     ; ((DE)) <- ((HL))
1A9A 12                 STAX    D
1A9B 23                 INX     H       ; INC SOURCE ADDR
1A9C 13                 INX     D       ; INC DEST ADDR
1A9D 0B                 DCX     B       ; DEC #CHRS
1A9E 78B1               MOV     A,B     !       ORA     C
1AA0 C2991A             JNZ     CMOV2   ; REPEAT IF #CHRS <> 0
1AA3 C9                 RET
                ;---------------------------------------
                ; (HL) < (DE)   ; PRE-DECREMENTING VERSION
1AA4 09         CMOV3   DAD     B       ; LAST SOURCE + 1
1AA5 EB                 XCHG
1AA6 09                 DAD     B       ; LAST DESTINATION + 1
1AA7 EB                 XCHG
1AA8 2B         CMOV4   DCX     H       ; DEC SOURCE ADDR
1AA9 1B                 DCX     D       ; DEC DEST ADDR
1AAA 7E                 MOV     A,M     ; ((DE)) <- ((HL))
1AAB 12                 STAX    D
1AAC 0B                 DCX     B       ; DEC #CHRS
1AAD 78B1               MOV     A,B     !       ORA     C
1AAF C2A81A             JNZ     CMOV4   ; REPEAT IF #CHRS <> 0
1AB2 C9                 RET
1AB3                    END
```

Wydruk 3

Na wydruku 2 przedstawiony został również sposób zainstalowania nowej wersji procedury **FORGET.** Wersja 2.1 M wykrywa następujące błędy:

ERROR # 24 gdy słowniki **CONTEXT** i **CURRENT** nie są równe,

ERROR # 21 gdy słowo jest zabezpieczone przed zniszczeniem przez odpowiednią wartość zmiennej **FENCE**,

ERROR # 14 gdy następuje próba skasowania słownika FORTH (nawet po zmianie wartości FENCE),

ERROR # 13 gdy następuje próba skasowania słownika **CONTEXT**.

Jak to już zostało zasygnalizowane, standardowa procedura **CMOVE** nie kopiuje prawidłowo nakładających się bloków pamięci. Nie jest to błąd; standard FORTH 79 przewiduje dwa słowa do przesyłania blokowego: słowo **CMOVE** przesyłające bajty od początku bloku (od najmniejszego adresu) oraz słowo **<CMOVE**, które przesyła

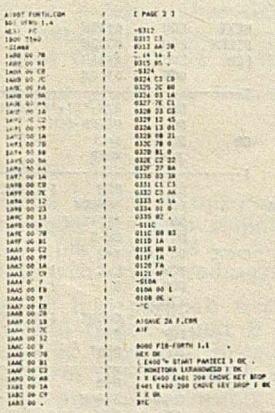
---

[1]) Dokumentacja fig-FORTHA 1.1 nie ogranicza wartości „n" do liczb dodatnich, jednakże nieparzyste wartości ujemne tego parametru powodują pojawianie się błędnych wydruków. Wyświetlanie komunikatu zerowego wymusza następująca sekwencja:

[DECIMAL 2 ' MESSAGE 12 + !
' DUP ' CFA ' MESSAGE 8 + !

[2]) Zgodnie z przyjętą przez FIG numeracją wersje 1.1 oraz FORTHDOS 1.2 nie mają zmian w jądrze języka — od LIT do TASK — przypis autora

bajty od największego adresu w kierunku adresów malejących (od końca bloku). W wersji 2.1 M przyjęta została zasada automatycznego określenia kierunku przesyłania blokowego. Analogiczne rozwiązanie zostało przedstawione na wydruku 3 w wersji INTEL 8080, przy czym ze względu na brak miejsca w pamięci w jądrze fig-FORTHA 1.1 część procedury została dopisana po słowie TASK. Przykładową sesję tworzenia nowej wersji języka przedstawia wydruk 4.



Wydruk 4

W odcinku sprzed dwóch miesięcy zostały opublikowane m. in. procedury VLIST, ?SPACES i TAB. Ponieważ nie zostało to wystarczająco podkreślone, pragnę wyjaśnić, iż żadna z tych procedur nie będzie działać prawidłowo, jeśli słowo CR nie będzie zerować zmiennej OUT. Odpowiednie zmiany podaje wydruk 3, a sposób ich generowania — wydruk 4.

Jak widać z wydruku 4, przy tworzeniu nowych wersji i języka z pliku CP/M typu .COM obowiązują następujące zasady:

— zmienia się wartości początkowe dla zmiennych FENCE (HEX 110), DP (11E) i VOC-LINK (120), przy czym dla DP przyjmuje się ostatnią wartość HERE (koniec słownika, z definicji : HERE DP @ ;),

```
                ;       SUBROUTINE USED BY - AND <
                ;                       ; (HL) <- (HL) - (DE)
083C                    ORG     083CH
083C 7D         SSUB    MOV     A,L     ; LB
083D 93                 SUB     E
083E 6F                 MOV     L,A
083F 7C                 MOV     A,H     ; HB
0840 9A                 SBB     D
0841 67                 MOV     H,A
0842 C9                 RET
                ;-------------------------------------------
0144 =          HPUSH   EQU     0144H
0861 =          LESS    EQU     0861H
087E                    ORG     087EH
087E 82                 DB      82H     ; U< ( UNSIGNED < )
087F 55                 DB      'U'
0880 BC                 DB      '<'+80H
0881 5D08               DW      LESS-4
                ;ULESS  DW      DOCOL,TDUP
                ;       DW      XORR,ZLESS
                ;       DW      ZBRAN,ULES1-$   ; IF
                ;       DW      DROP,ZLESS
                ;       DW      ZEQU
                ;       DW      BRAN,ULES2-$
                ;ULES1  DW      SUBB,ZLESS      ; ELSE
                ;ULES2  DW      SEMIS           ; ENDIF
                ;-------------------------------------------
0883 9508       ULESS   DW      $+2
0885 D1                 POP     D       ; (DE) <- (S1) = Y
0886 E1                 POP     H       ; (HL) <- (S2) = X
0887 CD3C08             CALL    SSUB    ; (HL) <- X - Y
088A 210000             LXI     H,0
088D D29108             JNC     ULES1
0890 2C                 INR     L
0891 C34401     ULES1   JMP     HPUSH
0894                    END
```

Wydruk 5

— jeżeli dopisane zostały nowe definicje, należy zmienić wartość słowa o adresie HEX 10C na adres pola NF ostatniego słowa w słowniku FORTH,

— na dysku należy zapisać zawartość pamięci od adresu 100_H do HERE.

Na zakończenie bieżącego odcinka zostanie zaprezentowana nowa wersja procedury U< napisanej w kodzie maszynowym (17 bajtów, bez nagłówka definicji — wydruk 5), a nie jak oryginalna — w języku FORTH (28 bajtów). Porównanie adresów nową procedurą trwa ok. 900 mikrosekund (mikroprocesor 8080, zegar 2 MHz), podczas gdy wykonanie starej wersji — sześć razy dłużej. W podane czasy wliczony jest czas pracy interpretera wewnętrznego („NEXT").

MICHAŁ CHOROSZUCHA
Warszawa

Wprawdzie zajęcia Akademii uległy oficjalnemu zakończeniu, to jednak zapraszamy Czytelników na wykład uzupełniający. O ile w pismach rozmaitego pokroju wprost zaroiło się od przeróżnych kursów programowania, to jednak znikoma liczba autorów wpaja swoim czytelnikom zasadę, że wiedza o tym, co realizuje taka czy inna instrukcja, to dopiero połowa (a może i mniej) umiejętności składającej się na sztukę programowania. O tym, że programowanie jest sztuką, przekonują się natychmiast wszyscy ci, którzy z jakichś względów zmuszeni są do wprowadzenia zmian w cudzych programach. O ile przy krótkich programach hołdowanie zasadzie: lepiej napisać własny program niż „przegryzać się" przez cudzy — może jeszcze dawać pewne rezultaty, to już przy tylko nieco dłuższych jest to praktycznie niemożliwe. Gorzej, gdy przychodzi „przegryzać się" przez ... własny program, tyle że napisany np. rok temu.

Krótki kurs Akademii mikroKLANU z pewnością nie jest w stanie wyczerpać całokształtu zagadnień związanych z umiejętnością „eleganckiego" programowania. Poniższe post scriptum poświęcone jest metodologii tworzenia programów średniej długości.

Post scriptum Akademii mikroKLANU czyli jak pisać duże programy

Tak się złożyło, że wszystkie odcinki Akademii mikroKLANU poświęcone były małym programom komputerowym. Aby zakończyć cykl, trzeba jeszcze wspomnieć o dużych programach. Wprawdzie większość użytkowników mikrokomputerów nigdy nie napisze naprawdę dużego programu (typu procesora tekstu lub systemu operacyjnego), ale nieco dłuższe programy zdarza się pisać prawie każdemu; wiele zastosowań mikrokomputera sprowadza się do napisania nietypowego programu. Programu takiego nie da się skopiować ani przerobić, trzeba go niestety napisać. Jak to zrobić skutecznie, postaram się pokazać w tym artykule. Jako przykład wybrałem program, który pisałem dla znajomego. Spełnia on opisane powyżej założenia, gdyż jest programem „średniodługim" (ok. 300 linii, ale nie będziemy ich wszystkich reprodukowali), a jednocześnie był pisany z myślą o bardzo prostym komputerze (ZX81) oraz o zupełnie niewprawnym użytkowniku, dla którego wiele wewnętrznych możliwości komputerka jest jeszcze tajemnicą. Program umożliwia sterowanie (poprzez specjalny interfejs) elektroniczną perkusją i służy do programowanego zmieniania granych melodii. Takie urządzenia oczywiście można już kupić (w cenie ok. 400 dol. USA), ale mój znajomy woli je zrobić sam (w Polsce ciągle łatwiej jest zbudować nawet skom-

plikowane urządzenie, niż zarobić pieniądze na jego kupno...) [1]).

Jesteśmy więc na starcie: postawiono problem, a my myślimy, jak może on zostać rozwiązany z wykorzystaniem mikrokomputera. Ta faza projektowania programu jest bardzo ważna. Można powiedzieć, że od właściwych decyzji na tym etapie zależy powodzenie całego projektu. A więc nie spieszmy się i dobrze przemyślmy cały problem. Zacznijmy od zanotowania, co program ma robić. W naszym przykładzie chcemy, by komputer zapamiętał 16 melodii, przy czym każda melodia składa się z 32 nut (część z nich może się powtarzać) Mamy do dyspozycji 12 instrumentów, ale prawdopodobnie nigdy wszystkie nie będą wykorzystane w jednej melodii. Należy więc zrobić zestawienie częstości wykorzystania poszczególnych instrumentów: tu pomoże nam ktoś doświadczony w sprawach muzycznych. Jeżeli zastanowimy się nad mnogością melodii oraz instrumentów, to dojdziemy do wniosku, że trzeba przyjąć jakiś sposób kodowania tak dużej liczby informacji. Jako wprawkę piszemy więc procedurę kodującą użycie poszczególnych instrumentów w danej nucie. Wydaje się, że kodowanie binarne jest tu optymalne. Oto proste procedury kodujące i dekodujące:

```
11000 REM PROCEDURA KODUJACA
11010 A=0
11020 FOR I=1 TO 12
11030 A=A+N(I)*2**I
11040 NEXT Y
11050 RETURN


10000 REM PROCEDURA DEKODUJACA
10010 IF A/2()INT(A/2) THEN GOTO 10100
10020 FOR I=12 TO 1 STEP -1
10030 N(I)=0
10040 B=A-2**I
10050 IF B<0 THEN GOTO 10080
10060 N(I)=1
10070 A=B
10080 NEXT I
10090 RETURN
10100 PRINT "BLAD DANYCH!"
10110 STOP
```

Ponieważ ZX81 ma niezbyt pewny system nagrywania danych, nasz sposób kodowania ma jeszcze jedną zaletę: umożliwa sprawdzenie, czy wczytane dane nie zawierają błędu. Oczywiście w praktyce nie musimy od razu pisać procedur w sposób formalny. Wystarczy, jeżeli wiemy, jak będziemy postępowali, czyli określić algorytm.

Problem, z którym przyszło nam się zmierzyć, byłby łatwiejszy do rozwiązania w języku bardziej sformalizowanym niż BASIC (np. PASCAL), ale i tu możemy zastosować elementy programowania strukturalnego. Trzeba jednak pamiętać o globalnym charakterze zmiennych w języku BASIC.[2]) Dlatego też niezbędne jest narysowanie dokładnego schematu logicznego programu. Należy zacząć od bardzo ogólnie sformułowanych zadań dla poszczególnych bloków funkcjonalnych, a potem schodzić do coraz niższych modułów. Takie postępowanie nazywa sie programowaniem zstępującym i należy je polecić wszystkim piszącym dłuższe programy. Prosze zwrócić uwagę, że po przyjęciu takiego stylu programowania samo zapisywanie programu jest już niewielkim, finalnym zadaniem. Ze względu na globalny charakter zmiennych w języku BASIC, procedury należy wprowadzać tylko wtedy, gdy wielokrotnie powtarza się konieczność realizacji identycznego zadania, tzn. gdy dany ciąg instrukcji może być wywoływany przez program główny wiele razy. W naszym przykładzie na pewno taka procedura będzie rysowanie graficzne taktu 32-nutowego:

Jest to procedura jedna z najważniejszych w tym konkretnym zagadnieniu. Ponieważ jest ona bardzo długa — a w tym tekście raczej chodzi nam o pokazanie metodologii programowania — więc nie będziemy jej publikować.

Analiza schematu logicznego wykazała, że w każdym przypadku, tzn. gdy tworzymy nową melodię, gdy uzupełniamy starą lub gdy ją odtwarzamy, wygodnie jest zobaczyć na ekranie komputera rozkład użycia instrumentów (znana zasada „rysunek wart jest 1000 słów" przyjmuje w przypadku komputerów formę „...wart jest 1024 bajty"!).

Specjalnie nie zamieszczam schematu logicznego przykładowego programu, gdyż chciałbym, by Czytelnicy sami zastanowili się, jak taki schemat mógłby wyglądać. Gdyby ktoś miał wątpliwości, to w dalszym ciągu pokazany będzie program główny, skąd Czytelnicy będą mogli zorientować się, jakie możliwości były rozpatrywane. Pamiętajmy, że dobry schemat logiczny to także połowa (a może więcej) dokumentacji programu. A dobrze napisany program musi być udokumentowany. Prawdopodobnie nasz program będziemy wielokrotnie przerabiali, a więc dokładny opis, co robi każdy moduł, jaka jest rola zmiennych itp., będzie niezbędny. Wreszcie, w naszych notatkach powinny się znaleźć uwagi na temat testowania programu i jego rezultatów. W ten sposób będziemy mogli łatwo usprawnić program.

Jeżeli już dokonaliśmy logicznego podziału programu na małe moduły, warto sprawdzić, czy są one zrozumiałe, to znaczy, czy nie trzeba będzie w nich zastosować instrukcji GO TO. Niestety, BASIC narzuca nam tu znaczne ograniczenie: na ogół nie dysponujemy instrukcją ELSE, a więc nie będzie można zrezygnować z instrukcji GO TO. Warto jednak (mówiliśmy już o tym w Akademii) stosować pewne chwyty programowe, by w jednym wyrażeniu arytmetycznym wykonywać wiele wariantów (przypominam program suwania kursorem).

Mając już określoną strukturę modułów, możemy teraz przystąpić do zapisywania programu. Dobrze jest zrobić sobie podręczną listę zmiennych i trzymać się ustalonej konwencji (pamiętamy, że w programach Akademii oznaczaliśmy zmienną operacyjną literą X, stałe literami A, B, C itd.). Oto jak zakodowaliśmy nasz program główny:

```
10 REM PROGRAM "PERKUSJA"
20 DIM M(16,32),M$(16,10),N$(12,10),N(12),T(16)
30 PRINT "MENU:"
40 PRINT ""
50 PRINT "NOWA MELODIA   1"
60 PRINT "UZUPELNIENIE   2"
70 PRINT "ODTWORZENIE    3"
80 PRINT ""
90 INPUT "PODAJ KOD:";P
100 CLS
110 PRINT "MELODIE:"
120 FOR I=1 TO 16
130 IF M$(I)="0 THEN GOTO 140
140 PRINT I;"-";M$(1)
150 NEXT I
160 PRINT ""
170 INPUT "WYBIERZ MELODIE:";N
180 IF P)1 THEN GOTO 190
190 INPUT "NAZWA MELODII:";M$(N)
200 CLS
210 GOSUB 5000
220 PRINT M$(N)
230 GOSUB 6000
240 GOSUB 7000
250 IF P$()"KONIEC" THEN GOTO 230
260 INPUT "POWTORZYC 16 NUT (T/N)";P$
270 IF P$="T" THEN GOSUB 8000
280 INPUT "TEMPO:";T
290 GOSUB 9000
300 INPUT "POPRAWKI (T/N)";P$
310 IF P$="T" THEN GOTO 20
320 END
```

Dla większej przejrzystości programu oznaczyliśmy kodowaną melodię jako M, nazwy kodowanych melodii zapamiętamy w wektorze MS, nazwy instrumentów zapamiętamy w wektorze NS. Analiza problemu wykazała, że zamiast nut będziemy wyświetlali symbol danego instrumentu. Tak więc prócz nazw instrumentów dobrze jest pamiętać ich litery kodujące:

```
5000 REM PROCEDURA LADUJACA NAZWY INSTRUMENTOW
5010 N$(1)="S-BAS DRUM"
5020 N$(2)="T-TAM TAM"
5030 REM ITD.
5100 RETURN
```

Procedura 5000 może też wyświetlać pełne nazwy instrumentów wtedy, gdy użytkownik zapomni czym dysponuje. Takie podwójne wykorzystanie danych bardzo skraca program — ZX81 ma małą pamięć!

Chciałbym teraz zwrócić uwagę Czytelników na pewną trudność, jaką można napotkać w trakcie pisania programu. Otóż z podziału na moduły wynikło, że dwa bloki będą wywoływane po sobie (linie 220 i 230, gdzie procedura rysująca obrazek taktu po-

[1]) Nie zgadzam się z autorem. Mikrokomputer łatwiej i taniej jest kupić, natomiast zarobienie na niego to zupełnie inna sprawa, faktycznie w Polsce niełatwa. Trochę inaczej wygląda kwestia urządzeń specjalizowanych, które produkowane są w małych seriach. Rzeczywiście często opłaca się zrobić je samemu, tak jak urządzenie wspomniane w artykule (przyp. AJP)

[2]) Uwaga dla nowicjuszy: zmienna globalna to taka, która jest identycznie rozumiana w całym programie; w niektórych językach programowania można wprowadzić zmienne lokalne dla danej procedury i w ten sposób uniknąć błędów wynikających z przypadkowej zbieżności nazw zmiennych stosowanych w różnych procedurach (przyp. AJP)

przedza procedurę wczytującą kod instrumentu oraz pytającą, czy już koniec poprawek):

```
7000 REM PROCEDURA WCZYTUJACA KOD INSTRUMENTU
7010 REM DOKONUJE POPRAWEK W GRAFICE
7020 REM KORZYSTA Z PROCEDURY 11000
7100 RETURN
```

Można oczywiście zapytać, dlaczego nie połączyliśmy tych procedur. Ano, gdybyśmy chcieli zmodyfikować nasz program (a nastąpi to z całą pewnością!), wygodniej będzie mieć te procedury rozdzielone. Podobny argument odnosi się do procedury powielającej 16 nut (takt równo podzielony). Gdybyśmy nie zdefiniowali tej procedury, należałoby stosować skok bezwarunkowy, co bardzo zmniejsza przejrzystość programu (linia 260). A tak piszemy prostą procedurę-moduł:

```
8000 REM PROCEDURA POWIELENIE 16 NUT
8010 FOR I=1 TO 16
8020 M(N,I+16)=M(N,I)
8030 NEXT I
8040 RETURN
```

Wreszcie dochodzimy do celu: trzeba już tylko napisać właściwą procedurę sterującą interfejsu — najlepiej zrobić to w języku wewnętrznym, gdyż BASIC ZX81 jest stosunkowo wolny, a użytkownik może zażądać szybkiego grania melodii (nie możemy łudzić się, że nasza perkusja będzie wykorzystywana tylko do podawania taktu bardzo wolnego poloneza...).

```
9000 REM PROCEDURA W JEZYKU WEWNETRZNYM
9010 REM STERUJACA ELEKTRONICZNA PERKUSJA
9100 RETURN
```

Oczywiście wszystkie procedury przetestowaliśmy osobno i dopiero po upewnieniu się co do ich prawidłowości oraz poprawności operacji łączymy je w jeden, długi program.

Na zakończenie kilka uwag praktycznych, choć osobiście wierzę, że nic nie jest w stanie zastąpić doświadczenia (proszę Czytelników, by jako małą wprawkę spróbowali napisać program elektronicznej perkusji na swój, wybrany komputer!).

Przede wszystkim nie należy bać się pisania modułów od nowa. Jeżeli mamy kłopoty, jeżeli program wygląda nieelegancko lub jest niezbyt przejrzysty, to należy siąść i napisać go po raz drugi. Być może należy nawet cofnąć się do wcześniejszego etapu, tzn. do planowania i dobrze przemyśleć, czy problem postawiliśmy we właściwy sposób. Zwykle dobrze „rozgryziony" problem koduje się sam, natomiast jeżeli mamy wątpliwości, to lepiej jeszcze raz przeanalizować sytuację! Po drugie nie należy uciekać od testów. Tylko wszechstronne sprawdzenie programu (także przez przyszłego użytkownika!) zapewni nam dobrą jakość produktu finalnego. W naszym przykładzie dzięki modułowemu charakterowi programu oraz konsekwentnej regule użycia zmiennych udało się wprowadzić — już po zakończeniu pisania — wiele drobnych zmian, które wydały się użytkownikowi konieczne dla łatwiejszego operowania (np. wprowadzenie zmiennej liczby instrumentów). Dobra dokumentacja (czyli notatki, gdzie i jaka zmienna jest używana) pozwoliły na bezbolesne wprowadzanie kolejnych zmian, choć program ciągle się rozrastał. W pewnym momencie np. okazało się, że dotychczasowy podział na moduły należy zmienić dla przyspieszenia działania programu. I znowu przejrzysta dokumentacja ułatwi szybkie przeniesienie określonych operacji do programu głównego. Wszystkie tego rodzaju zmiany były po prostu nie do pomyślenia przed przetestowaniem programu. A wydawało się, że wszystkie możliwości zostały rozważone...

Wreszcie dochodzi się do momentu, gdy dalsze poprawianie programu jest ... bezcelowe — to znaczy czas poświęcony na szlifowanie drobiazgów nie opłaci się. Po prostu program będzie używany przez określony czas i wszystkie poprawki, które trwają dłużej niż czas używania programu, są nieekonomiczne (pamiętajmy, że program piszemy w celu wykonania zadania, a nie dla efektu estetycznego!). Wszystkim, którzy pasjonują się programowaniem, polecam maksymę: „Najlepszy program, to program działający"!

Zamykając ostatnie posiedzenie Akademii mikroKLANU życzę wszystkim Czytelnikom wielu działających programów!!

JAKUB TATARKIEWICZ

Żegnamy mikroKLAN na łamach INFORMATYKI

mikroKLAN towarzyszył Czytelnikom przez dwa roczniki INFORMATYKI. Przez ten czas przeżyliśmy prawdziwą „mikrorewolucję": mikrokomputery pojawiły się w Polsce nie tylko jako barwne fotografie z zagranicznych pism, ale trafiły nawet do prywatnych domów. Najważniejsze jednak, że stały się narzędziem pracy — wiele importowanych komputerów zostało zakupionych przez instytuty naukowe i wyższe uczelnie. Tańsze modele trafiły, na razie na prawach eksperymentu, do wybranych szkół średnich. Z wąskiej grupy hobbistów, do której początkowo adresowaliśmy mikroKLAN, wyrosła liczna rzesza użytkowników ZX SPECTRUM, COMMODORE C64, AMSTRADA, a ostatnio nawet MACINTOSHA i IBM-PC.

W tej sytuacji kontynuacja mikroKLANU jako wkładki do niskonakładowego, specjalistycznego pisma, które na domiar złego ukazuje się ostatnio z wielomiesięcznym opóźnieniem, nie ma większego sensu. Zeszyty INFORMATYKI stały się nawet przedmiotem spekulacji, a ograniczona objętość wkładki powodowała, że nie można było w niej omówić wielu ważnych zagadnień.

Z górą rok temu podjęliśmy starania o utworzenie osobnego pisma poświęconego zagadnieniom związanym z racjonalnym stosowaniem techniki mikrokomputerowej. Przez długi okres nikt nie chciał uwierzyć (oczywiście z wyjątkiem Czytelników mikroKLANU, którzy zarzucali nas listami, nie mówiąc o telefonach), że takie pismo jest potrzebne, że może być sprzedawane nawet w 100 tys. nakładzie.

Po przezwyciężeniu przeszkód, wynikających z deficytu papieru i możliwości druku, wydawnictwo SIGMA uzyskało zezwolenie na wydawanie w 1986 roku serii broszur o nazwie „Mikroklan".

Czytelników, którzy mając na uwadze trudności, jakie występowały z nabyciem INFORMATYKI, chcieliby zaprenumerować „Mikroklan" musimy na razie rozczarować: broszury — jak wszystkie wydawnictwa nieperiodyczne — nie zostaną objęte prenumeratą.

REDAKCJA

prowadzi
ANDRZEJ J. PIOTROWSKI
tel. dom.: 48-22-85